Comprando esta edição tenha acesso na WEB a:
+390 vídeo-aulas +150 artigos e 6 cursos online

www.clubedelphi.net **Feita para desenvolvedores**

ClubeDelphi

Ano 7 · Edição 86 · R$9,90

DevMedia GROUP

ISSN 1517990-7

SOAP, XML & Web Services

- **Construa aplicações em camadas com diferentes tipos de cliente (Win32, .NET e Mobile) que acessam o mesmo Web Service**
- **Use ASP.NET para construir uma interface Web que acessa o BD via SOAP**
- **Integre aplicações em D7 com .NET**
- **Crie aplicativos móveis (Pocket PC) para acessar o servidor e BD**

Brinde!

Na compra desta edição ganhe **6 vídeo-aulas** sobre Ferramentas para **turbinar o Delphi!**

- Comparação de arquivos fonte com Compare It
- Formatando código-fonte com o DelFor Exp
- Trabalhando com o Simple Resource Editor
- Conhecendo o plugin Gexperts
- Conhecendo o plugin CnWizards
- Compactação de arquivos com UPX

ASP.NET
Super dicas para turbinar o DataGrid: colunas template, imagens, master/detail e totalização

Web Services
Construa um disco virtual usando WS e técnicas avançadas de SOAP – Parte 2

Delphi 2007 em detalhes
Veja na prática os novos recursos para suporte ao Windows Vista, temas, gráficos e AJAX

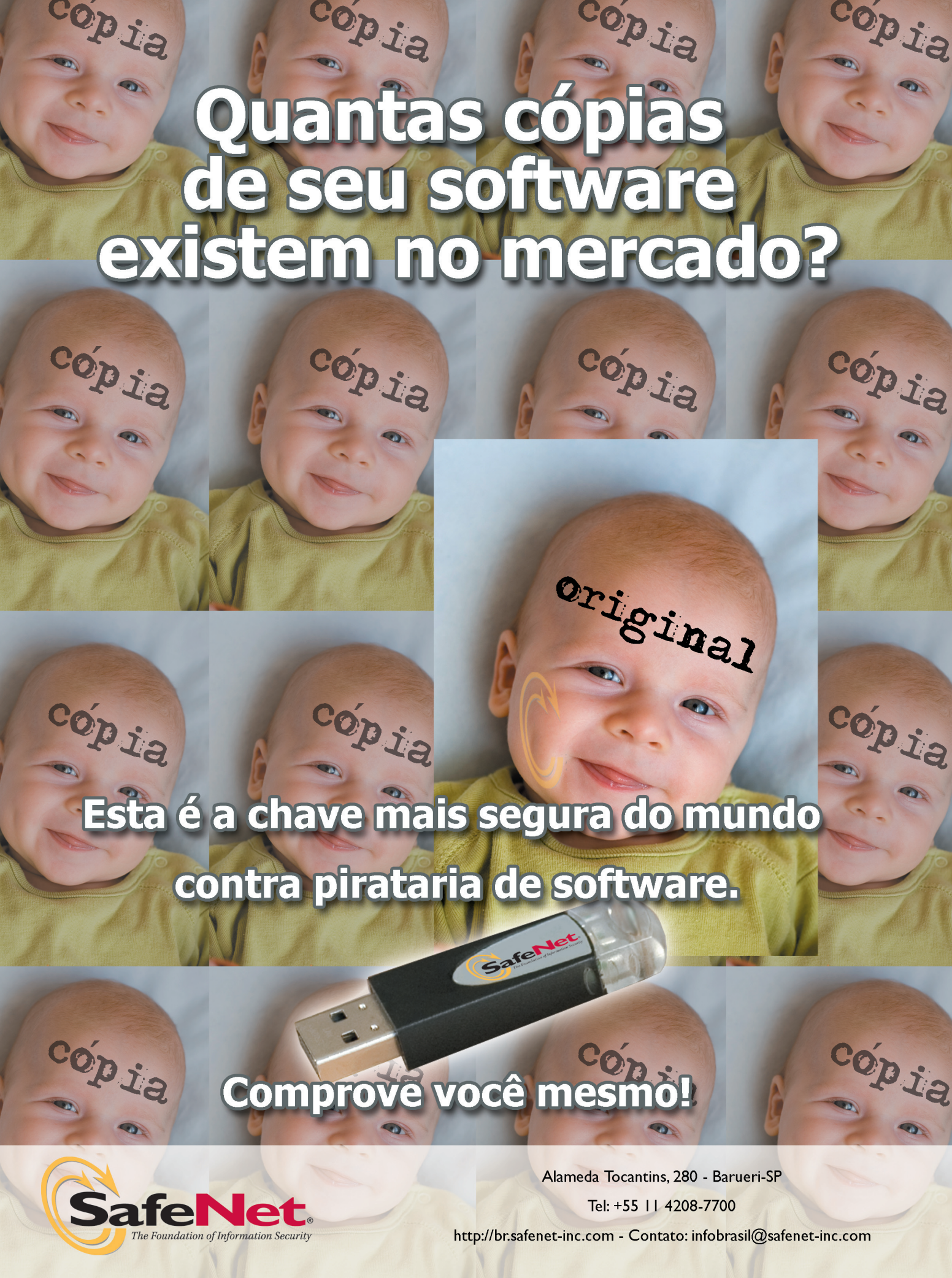
Boas práticas
Dicas para trabalhar em equipe de forma produtiva e controlada

Relatórios
Impressão matricial com Delphi

Mão na Massa
Publique seus relatórios do Quick Report na Web!

Coluna Ask the Expert

Quantas cópias de seu software existem no mercado?



Esta é a chave mais segura do mundo contra pirataria de software.

Comprove você mesmo!

SafeNet
The Foundation of Information Security

Alameda Tocantins, 280 - Barueri-SP

Tel: +55 11 4208-7700

http://br.safenet-inc.com - Contato: infobrasil@safenet-inc.com

ClubeDelphi

Ano 7 - 86ª Edição - 2007 - ISSN 1517990-7 **Impresso no Brasil**

Corpo Editorial

Diretor Editorial e Administrativo
Gladstone Matos
gladstone@neoficio.com.br

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Antonio Xavier
webdesigner@devmedia.com.br

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Luciano Pimenta
lucianopimenta@clubedelphi.net

Revisão
Rafael de Castro Santos
rafael@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor
A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Jeff Wendell
jeff@devmedia.com.br

ClubeDelphi PLUS

DevMedia GROUP

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS. Para mais informações sobre o pacote PLUS, acesse: *http://www.devmedia.com.br/clubedelphi/portal.asp*

Apoio

ASSESPRO PARANÁ

ÍNDICE

06 - Integrando aplicações .NET, Win32 e Mobile com SOAP e Web Services
Guinther Pauli

16 - Dicas de DataGrid
Luciano Pimenta

22 - Delphi 2007 em detalhes
Adriano Santos

28 - SOAP - Aprenda técnicas avançadas em um exemplo passo a passo
Michael Benford

38 - Impressão matricial no Delphi
Rodrigo Otto Mostaert

42 - Publique na Web seus relatórios do QuickReport
Fabrício Desbessel

46 - Boas práticas para trabalhar em equipe
Fernando Sarturi Prass e Cristiano Caetano

EDITORIAL

A integração entre sistemas é uma necessidade cada vez mais constante. Os Web Services permitiram que aplicações pudessem se comunicar através do SOAP (Simple Object Access Protocol), e desde o Delphi 6 podemos contar com esse suporte. Web Services e SOAP são baseados em XML, o que permite que aplicações rodando em plataformas heterogêneas possam se comunicar. E o melhor de tudo, através da Internet. Nesta edição, você vai aprender a construir uma solução multicamadas em Delphi baseada nessas tecnologias, onde um servidor Web Service vai concentrar e centralizar o acesso a dados e regras de negócio, que são consumidas por diferentes tipos de cliente: Web (ASP.NET), Desktop (Delphi 7 Win32) e Mobile (Compact Framework).

Ainda nesta edição, muitas dicas sobre o principal controle para desenvolvimento Web com ASP.NET e Delphi, o DataGrid, no artigo de Luciano Pimenta. O Adriano continua a série de artigos sobre a novíssima versão do Delphi, aprofundando temas como suporte ao Windows Vista e AJAX. SOAP também é o tema do artigo do Michael Benford, na segunda parte da aplicação que mostra como criar um disco virtual. O Rodrigo mostra como é simples a impressão matricial em Delphi. O Fernando Prass e o Cristiano dão boas dicas para você obter maior produtividade e controle no trabalho em equipe. E finalmente, o Fabrício apresenta uma solução inteligente para uma tarefa que parecia impossível, publicar na Web seus relatórios do Quick Report.

Um abraço a todos e muito sucesso com o Delphi!

Guinther Pauli
guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

Fale com o Editor

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

Luciano Pimenta - Editor do Site
lucianopimenta@clubedelphi.net

Portal do Assinante

NÃO PERCA

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:
- Mais de 390 Vídeo Aulas!
- 6 cursos online!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 140 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse *www.devmedia.com.br/clubedelphi/portal.asp* e utilize as informações abaixo: **Login: DVM.PL e Senha: CLB86**

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Ask The Expert

Perguntas e Respostas

Dúvidas respondidas por **Guinther Pauli**
(envie as suas para guinther@devmedia.com.br)

SubComponentes

Como alguns componentes da VCL, como o SimpleDataSet e LabeledEdit, conseguem ter subcomponentes associados com todas as propriedades acessíveis internamente? Gostaria de criar um componente com essas características mais não sei como.

Rodrigo Pizolotto

A partir da versão 6, o Delphi passou a suportar o recurso de *SubComponentes*. Como exemplo, vamos criar uma classe descendente de *ClientDataSet* que possui a característica de *AutoRefresh*, sendo que o intervalo de *Refresh* é controlado por um *Timer* interno que será usado com o recurso de subcomponentes. O código é mostrado na **Listagem 1**.

Para criar esse componente, crie um novo *Package* a partir do *Object Repository* e a seguir adicione um componente a ele. Quando usamos o componente em uma aplicação, o CDS dá um *refresh* automático a cada *n* segundos. Observe que a "mágica" toda está na chamada ao *SetSubComponent*. O resultado é mostrado na **Figura 1**.

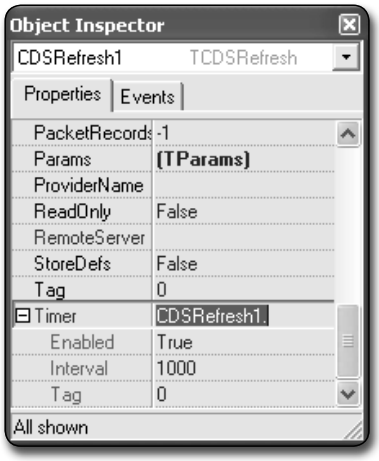


Figura 1. SubComponentes

Listagem 1. Exemplo de AutoRefresh

```
unit uCDSRefresh;
interface
uses
  SysUtils, Classes, DB, DBClient, ExtCtrls;
type
  TCDSRefresh = class(TClientDataSet)
  private
    FTimer: TTimer;
    procedure DoTimer(Sender: TObject);
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property Timer: TTimer read FTimer;
  end;
  procedure Register;

implementation
procedure Register;
begin
  RegisterComponents('Guinther', [TCDSRefresh]);
end;
constructor TCDSRefresh.Create(AOwner: TComponent);
begin
  inherited;
  if Assigned(FTimer) then exit;
  FTimer := TTimer.Create(self);
  FTimer.OnTimer := DoTimer;
  FTimer.SetSubComponent(True);
end;
destructor TCDSRefresh.Destroy;
begin
  FTimer.Destroy;
  inherited;
end;
procedure TCDSRefresh.DoTimer(Sender: TObject);
begin
   if Active then begin
     Close;
     Open;
   end;
end;
end.
```

ClientDataSet não exclui dados fisicamente

Estou trabalhando em uma aplicação que salva os dados em XML. Quando eu excluo um registro em um DBGrid, por exemplo, ele deixa de ser exibido em tela, mas quando abro o arquivo XML, verifico que mesmo excluído ele continua lá. O que o ClientDataSet faz?

Thiago Balbino

Olá Thiago. O *ClientDataSet* mantém atualizações em cache para poder trabalhar de forma correta com o *ApplyUpdates*. O *DataSetProvider* precisa desses valores excluídos, alterados, ou inseridos, para gerar instruções SQL de atualização.

No entanto, se você está trabalhando com XML local (também conhecido como MyBase), não precisará armazenar o *delta*. Para isso, basta chamar o método *MergeChangeLog* do *ClientDataSet* antes do salvamento.

Assinatura ClubeDelphi PLUS
Mais conteúdo .NET por menos!

+390 vídeo aulas | 6 cursos online

www.clubedelphi.net/portal

Caro Leitor,

O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 390 vídeo aulas; ii) 6 cursos online; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 150 artigos exclusivos (que não foram publicados na revista)!;
Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).
Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.
Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!
Equipe DevMedia

Curso Online - PLUS

O portal ClubeDelphi PLUS conta com 6 cursos online, confira!

Sistema completo com Delphi 7, dbExpress e Firebird 2.0 [*Luciano Pimenta*]

Aplicações WEB com IW e Delphi 7 (Delphi Win32)! [*Guinther Pauli*]

Aplicações client/server no Delphi 2006 [*Luciano Pimenta*]

Criando uma Aplicação multi-camadas Completa com Delphi [*Guinther Pauli*]

Criando uma aplicação completa: sistema SysCash [*Everson Volaco*]

Aplicações Client/Server com dbExpress e Firebird [*Luciano Pimenta*]

Últimas Vídeo-Aulas (Somente para assinantes)

Sistema completo com Delphi 7, dbExpress e Firebird 2.0
Acompanhe a criação de um sistema completo de gerenciamento de locadora, usando Delphi, dbExpress e Firebird 2.0. Luciano Pimenta, mostrará todos os passos necessários para a criação do projeto em todas as fases de implementação, desde a criação do banco de dados, até a migração para .NET.

Criando uma tela de consulta dinâmica – Parte IV e V
Veja nessa vídeo aula de Paulo Quicoli, como criar um tela de consulta dinâmica.

Mini curso de ClientDataSet - Parte I a VI
Veja nessas aulas de Guinther Pauli um mini curso exclusivo sobre dicas avançadas sobre o ClientDataSet.

MainMenu no Delphi for PHP
Veja nessa vídeo aula de Fabricio Desbessel como criar um menu no Delphi for PHP.

Trabalhando com Word no Delphi – Parte III
Veja nessa vídeo aula de Jefferson Junglaus, como trabalhar com o Word no Delphi.

Instalando o FastReport 4 no Delphi 2007 – Parte I e II
Veja nessa vídeo aula de Paulo Quicoli, como instalar a nova versão no FastReport no Delphi 2007.

Conhecendo o plugin GExpert
Veja nessa vídeo aula de Adriano Santos, como instalar e conhecer as características do plugin GExpert.

Conhecendo o plugin CnWizards
Veja nessa vídeo aula de Adriano Santos, como instalar e conhecer as características do plugin CnWizards.

.:: DESTAQUE ::.

Confira no portal ClubeDelphi PLUS 6 vídeo aulas sobre Ferramentas para trabalhar com o Delphi

Compactação de arquivos com UPX
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5393

Comparação de arquivos fonte com o Compare It
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5431

Formatando código-fonte com o DelFor Exp
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5454

Trabalhando com o Simple Resource Editor
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5476

Conhecendo o plugin GExperts
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5609

Conhecendo o plugin CnWizards
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5652

Integrando aplicações .NET, Win32 e Mobile com SOAP e Web Services

Guinther Pauli
guinther@devmedia.com.br)
Atua no ramo de tecnologia e programação há mais de 17 anos, é autor de mais de 100 artigos publicados, 200 vídeo-aulas e do livro "Delphi – Programação para Banco de Dados e Web". É Bacharel em Sistemas de Informação, Microsoft Certified: MCP, MCAD, MCSD.NET, Borland Certified: Delphi 6, 7, 2005, 2006, Web e Kylix. Editor Geral das Revistas .net Magazine, ClubeDelphi, WebMobile (.NET) e Mr.Bool. Palestrante em vários eventos pelo Brasil, como TechDay SP,RJ,POA,BH, Web Days e todas as edições da Borland Conference.

Na edição 70 da ClubeDelphi, propus uma solução baseada em SOAP, XML e *ClientDataSet* para permitir que aplicações se comunicassem através de Web Services. Entre algumas vantagens da solução, destacaria o acesso ao BD a partir de qualquer lugar da Internet e centralização do acesso a dados em uma única camada.

A aplicação cliente Win32 poderia, por exemplo, ficar em um determinado país e obter dados via Web Services de um servidor rodando em um outro canto do globo. Tudo trafegado através de XML sobre o HTTP. São as facilidades da distribuição Web aliadas ao RAD para Desktop.

Neste artigo vamos dar um passo a mais nessa proposta. Além de permitir todas essas vantagens já citadas, vamos adicionar mais um elemento importante na arquitetura: nossa mesma solução poderá ter clientes rodando em diferentes plataformas.

Nossa solução será, obviamente, multi-camadas. Teremos uma camada servidora, na forma de um Web Service, que se encarregará de acessar o banco de dados. Diferentes tipos de aplicação farão acesso a esse Web Service, solicitando dados e chamando métodos para atualizar informações.

Arquitetura da solução

Para ter uma visão geral da arquitetura da aplicação que será criada, veja o esquema da **Figura 1**. Durante todo este tutorial, sempre que tiver uma dúvida de como a solução está sendo implementada e em que camada está sendo empregada determinada tecnologia, consulte essa figura. Ela será nosso "mapa".

O Web Service será criado em Delphi for .NET e utilizará ADO.NET para comunicação com o banco de dados (SQL Server ou Firebird para este exemplo). Um cliente ASP.NET será a interface Web da solução, permitindo que a aplicação seja acessada por qualquer browser pela Internet.

Uma aplicação Win32 em Delphi 7 acessará os mesmos dados, comunicando-se com o Web Service através de SOAP pela Intranet ou Internet. E finalmente, uma aplicação Mobile construída com Delphi for .NET e Compact Framework vai permitir acesso aos mesmos dados a partir de Pocket PCs.

Usando as tecnologias abordadas neste artigo, você poderá ao final:

• Acessar seu banco de dados a partir de qualquer lugar da Internet, usando qualquer um dos três tipos de cliente. Mesmo a solução Desktop pode se conectar ao BD remotamente através da Internet, graças ao SOAP;

• Criar aplicações distribuídas para a Internet. Você pode, por exemplo, centralizar o banco de dados e servidor de aplicação na matriz da empresa e distribuir as aplicações clientes para as filiais, que terão acesso a dados em tempo real e que se mantêm sempre atualizados, sem a necessidade de sincronização;

• Poderá ter sua aplicação oferecida como um serviço: seus clientes podem acessar a aplicação através da Internet, sem a necessidade de nenhuma configuração local na empresa, tudo fica configurado e armazenado no servidor;

• Centralizar todo o acesso a dados em uma única camada servidora, que também contém as regras de negócio. Os aplicativos clientes compartilham essa mesma camada, garantindo independência de banco de dados e tecnologia de acesso. Por exemplo, uma única modificação em um SQL ou código de negócio deve ser feito apenas no servidor e automaticamente é refletido nas aplicações clientes;

• Oferecer ao seu cliente três tipos de acesso ao servidor. Poderá utilizar a interface Web para acesso via browser. A aplicação desktop pode ser usada tanto em ambiente Intranet quanto Internet. E finalmente, os clientes podem usar dispositivos móveis como Pocket PCs para acessarem o mesmo servidor e BD.

Criando o banco de dados

O primeiro passo será a criação do banco de dados. Para este exemplo, vou utilizar o banco de dados free SQL Server 2005 Express, que pode ser obtido no endereço indicado na seção *Links*. Nesse mesmo endereço você encontra para download o *SQL Server Management Studio Express*, front-end para trabalhar com o SQL Server 2005.

Usando o *Management Studio*, crie um novo banco de dados chamado CUSTOMERS e dentro dele crie uma nova tabela também chamada CUSTOMERS. A estrutura dessa tabela deve seguir o modelo mostrado na **Tabela 1.** Lembre-se de definir o campo chave como *Identity* para que os valores sejam criados automaticamente.

Note que vamos criar uma tabela extremamente simples, para que possamos focar nossas atenções nos aspectos relativos à multicamadas, Web Services e multi-plataforma. Caso queira usar um banco de dados maior e pronto, como o *Northwind* ou *Employee*, fique a vontade.

A solução demonstrada neste exemplo funciona perfeitamente com o Firebird, caso queira utilizar esse banco de dados. No endereço para download deste artigo, você encontra uma versão do mesmo aplicativo deste artigo usando o banco de dados open source.

Após criado o banco de dados, insira alguns registros na tabela CUSTOMERS para facilitar nossos testes quando formos construir a aplicação no Delphi.

Construindo o servidor

Essa será a camada responsável por toda a comunicação com o banco de dados. Por ser baseada em XML Web Services, essa camada poderá ser acessada por diferentes tipos de aplicação rodando em diferentes plataformas. Além disso, esse acesso não precisa necessariamente ser feito em uma LAN, pois dados em XML e chamadas a métodos SOAP podem ser feitos através da Internet.

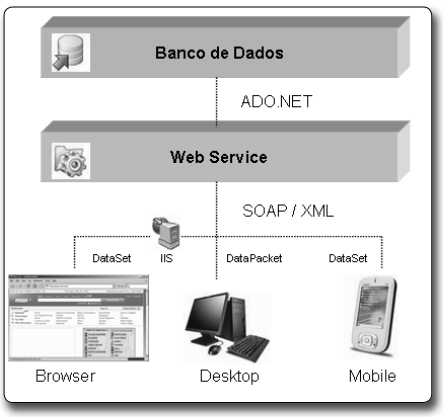


Figura 1. Arquitetura da solução

No Delphi, clique em *File>New>Other* e na categoria *Delphi for .NET Projects* escolha *ASP.NET Web Service Application*. Dê o nome de "DataServer" para a nova aplicação e salve o arquivo ASMX como "customer.asmx", alterando o seu nome no *Project Manager*. O arquivo ASMX é o que define o nosso Web Service. Cada classe dentro de um ASMX representa uma tabela do banco de dados nessa solução que estou propondo.

> **Nota**: Na arquitetura proposta, o Web Service é a camada que será encarregada do acesso e persistência dos dados, camada também conhecida como DAL (Data Access Layer).

Acesse o código do serviço, clicando na aba *customer.pas* logo abaixo no editor, e renomeie a classe criada para "TCustomer". Lembre-se de fazer essa alteração em todo o código já criado. Se quiser, pode usar o recurso de *SynchEdit* para alterar todas as referências de uma única vez, bastando para isso selecionar todo o código, clicar no botão e fazer a alteração (use o TAB).

Métodos para manutenção

Nossa classe *TCustomer* vai prover os métodos necessários para manipular os dados da tabela CUSTOMERS. Vamos começar implementando as operações básicas de manutenção de clientes.

Declare os métodos da **Listagem 1** na seção *public* da classe. Observe que os métodos precisam ter o atributo *WebMethod* para poderem ser acessados através de SOAP. Aperte *Shift+Ctrl+C* e implemente os métodos conforme mostrado na **Listagem 2.**

Estamos usando ADO.NET para comunicação com o SQL Server. O namespace *System.Data.SqlClient* deve ser importado para termos acesso às classes necessárias. Em ambos os métodos, criamos um objeto de conexão (*SqlConnection*) e utilizamos um *SqlCommand* para executar um determinado comando.

Coluna	Tipo
CustomerId	Integer (chave)
FirstName	Varchar(50)
Company	Varchar(50)

Tabela 1. Campos da tabela CUSTOMERS

O *ExecuteNonQuery* executa o comando de *update* ou *insert*. Para simplificar eu omiti o método *delete*, ele é bastante semelhante aos demais e pode ser facilmente construído.

O fechamento da conexão está protegido por um bloco *try finally* para garantir que a conexão seja devolvida para o *pool* de conexões mesmo que um erro aconteça. *Connection Pooling* é o mecanismo que permite ao ADO.NET reaproveitar conexões ao banco de dados, garantindo performance.

Caso esteja utilizando Firebird, você deverá utilizar os componentes nativos para acesso via ADO.NET. Para mais informações de como fazer a instalação, consulte a edição 66. Feita a instalação, referencie o assembly *FirebirdSql.Data.Firebird.dll* através da opção *Add Reference* no *Project Manager* e adicione o namespace no *uses*. O código muda pouco, basta usar *FbCommand* no lugar de *SqlCommand*, *FbConnection* no lugar de *SqlConnection* e assim por diante.

A função *ConnectionString* está declarada na seção *private* da classe e tem a implementação mostrada na **Listagem 3.** A *ConnectionString* é usada como parâmetro no construtor da classe de conexão. Uma opção seria armazenar essa *string* no *Web.Config*. Se estiver usando o Firebird, note que a mesma tem parâmetros um pouco diferentes.

Método para seleção

O método que obtém dados é chamado *SelectDS* (o "DS" é de "DataSet", pois esse é o tipo de retorno). Diferente dos métodos de *update* e *insert*, esse método precisa retornar o conjunto de dados obtido através da execução do comando *select*. No ADO.NET, um conjunto de dados pode ser representado basicamente de duas formas: por um *DataReader* ou por um *DataSet*.

Como os dados vão precisar ser trafegados pela Web em formato XML e o cliente precisa trabalhar de forma desconectada do BD (off-line), vamos usar um *DataSet*, já que o *DataReader* exige uma conexão e não pode ser serializado.

O código da **Listagem 4** mostra a implementação do método *SelectDS*. Aqui temos uma nova classe, um *SqlDataAdapter* que é utilizado para preencher os dados no *DataSet* através do método *Fill*.

A função que criamos recebe um parâ-

Listagem 1. Definição dos métodos SOAP

```
[WebMethod]
procedure Update(CustomerID: integer;
  Name,Company: string);
[WebMethod]
procedure Insert(Name, Company: string);
```

Listagem 2. Implementação dos métodos de manutenção

```
uses System.Data.SqlClient;
...
procedure TCustomer.Update(
  CustomerID: integer; Name, Company: string);
var
  Con: SqlConnection;
  Cmd: SqlCommand;
  SQL: string;
begin
   Con := SqlConnection.Create(ConnectionString);
   SQL := System.&String.Format(
     'update CUSTOMERS  '+'set FirstName=''{1}'', Company=''{2}'' '+
     'where CustomerId={0}', [CustomerId,Name,Company]);
   Cmd := SqlCommand.Create(SQL,Con);
   Con.Open();
   try
     Cmd.ExecuteNonQuery();
   finally
     Con.Close();
   end;
end;

procedure TCustomer.Insert(Name, Company: string);
var
  Con: SqlConnection;
  Cmd: SqlCommand;
  SQL: string;
begin
   Con := SqlConnection.Create(ConnectionString);
   SQL := System.&String.Format(
     'insert into CUSTOMERS (FirstName,Company) '+' values (''{0}'',''{1}'')',[Name,Company]);
   Cmd := SqlCommand.Create(SQL,Con);
   Con.Open();
   try
     Cmd.ExecuteNonQuery();
   finally
     Con.Close();
   end;
end;
```

Listagem 3. Método que retorna a ConnectionString

```
function TCustomer.ConnectionString: string;
begin
  Result := 'data source=".\SQLEXPRESS";'+
    'integrated security=True;'+'initial catalog=CUSTOMERS';
end;
```

Listagem 4. Executando um select e retornando um DataSet

```
public
  [WebMethod]
  function SelectDS(Filter: string): DataSet;
...
function TCustomer.SelectDS(Filter: string): DataSet;
var
  Con: SqlConnection;
  Adpt: SqlDataAdapter;
  Cmd: SqlCommand;
  Dst: DataSet;
  SQL: string;
begin
  if Filter = '' then Filter := '1=1';
  Con := SqlConnection.Create(ConnectionString);
  SQL := 'select * from CUSTOMERS where ' + Filter;
  Cmd := SqlCommand.Create(SQL,Con);
  Adpt := SqlDataAdapter.Create(Cmd);
  Dst := DataSet.Create;
  Adpt := SqlDataAdapter.Create(Cmd);
  Adpt.Fill(Dst);
  Result := Dst;
end;
```

metro que é a condição *where* do *Select*. Note também que estamos retornando todos os campos, caso queira você pode criar um outro método que retorne somente os campos que desejar.

Método para Update de DataSet

Os métodos *update* e *insert* que codificamos anteriormente recebem os valores a serem atualizados diretamente por parâmetro. Quando trabalhamos com ADO.NET, podemos usar um *SqlDataAdapter* para obter atualizações de um *DataSet* e gerar os respectivos comandos de *update*, semelhante ao que faz um *DataSetProvider* no *DataSnap*.

Dessa forma, criei mais dois métodos (**Listagem 5)** que processam atualizações via *DataSet* e *SqlDataAdapter*, que podem ser utilizados diretamente por aplicações cliente que sejam construídas sobre o .NET Framework e reconheçam a classe *DataSet*.

Por exemplo, uma aplicação ASP.NET vai receber um *DataSet* com dados de uma consulta, vai guardar em cache, processar atualizações nesses dados e finalmente vai devolver o *DataSet* alterado para o Web Service, que enviará as alterações ao BD, através desses métodos.

Uma opção aqui, caso queira, é codificar um único método que processe a inserção e atualização, também é possível.

Você já pode executar o Web Service e fazer alguns testes. O .NET Framework gera por padrão uma página para teste do Web Service, usando *Reflection* para obter informações sobre os métodos e criando a UI necessária para preencher os parâmetros para os métodos (**Figura 2**).

Faça alguns testes para o *update* e *insert* e verifique que as alterações nos dados realmente foram efetivadas no BD. Para o *select*, veja que os dados são retornados em um documento XML, o que comprova que um *DataSet* pode facilmente ser trafegado via HTTP.

Nossa camada servidora não contém interface (a da **Figura 2** é apenas para teste e não tem finalidade para o usuário final), apenas cuida do acesso a dados e regras

Listagem 5. Métodos para atualização de DataSets

```
public
  [WebMethod]
  procedure InsertDS(Dst: DataSet);
  [WebMethod]
  procedure UpdateDS(Dst: DataSet);
...
procedure TCustomer.InsertDS(Dst: DataSet);
var
  Con: SqlConnection;
  Adpt: SqlDataAdapter;
  Cmd: SqlCommand;
  SQL: string;
begin
  Con := SqlConnection.Create(ConnectionString);
  SQL := 'insert into CUSTOMERS ' + '(FirstName,Company) values ' +
  '(@FirstName, @Company);';
  Cmd := SqlCommand.Create(SQL,Con);
    Cmd.Parameters.Add(SqlParameter.Create('@CustomerId',SqlDbType.Int,4,'CustomerId'));
  Cmd.Parameters.Add(SqlParameter.Create('@FirstName',SqlDbType.VarChar,50,'FirstName'));
  Cmd.Parameters.Add(SqlParameter.Create('@Company',SqlDbType.VarChar,50,'Company'));
  Adpt := SqlDataAdapter.Create();
  Adpt.InsertCommand := Cmd;
  Adpt.Update(Dst);
end;

procedure TCustomer.UpdateDS(Dst: DataSet);
var
  Con: SqlConnection;
  Adpt: SqlDataAdapter;
  Cmd: SqlCommand;
  SQL: string;
begin
  Con := SqlConnection.Create(ConnectionString);
  SQL := 'update CUSTOMERS set ' + 'FirstName = @FirstName, ' +
  'Company = @Company '+
  'where CustomerId = @Original_CustomerId';
  Cmd := SqlCommand.Create(SQL,Con);
  Cmd.Parameters.Add(SqlParameter.Create(
    '@Original_CustomerId',SqlDbType.Int,4, 'CustomerId'));
  Cmd.Parameters.Add(SqlParameter.Create('@CustomerId',SqlDbType.Int,4,'CustomerId'));
  Cmd.Parameters.Add(SqlParameter.Create('@FirstName',SqlDbType.VarChar, 50, 'FirstName'));
  Cmd.Parameters.Add(SqlParameter.Create('@Company',SqlDbType.VarChar,50,'Company'));
  Adpt := SqlDataAdapter.Create();
  Adpt.UpdateCommand := Cmd;
  Adpt.Update(Dst);
end;
```

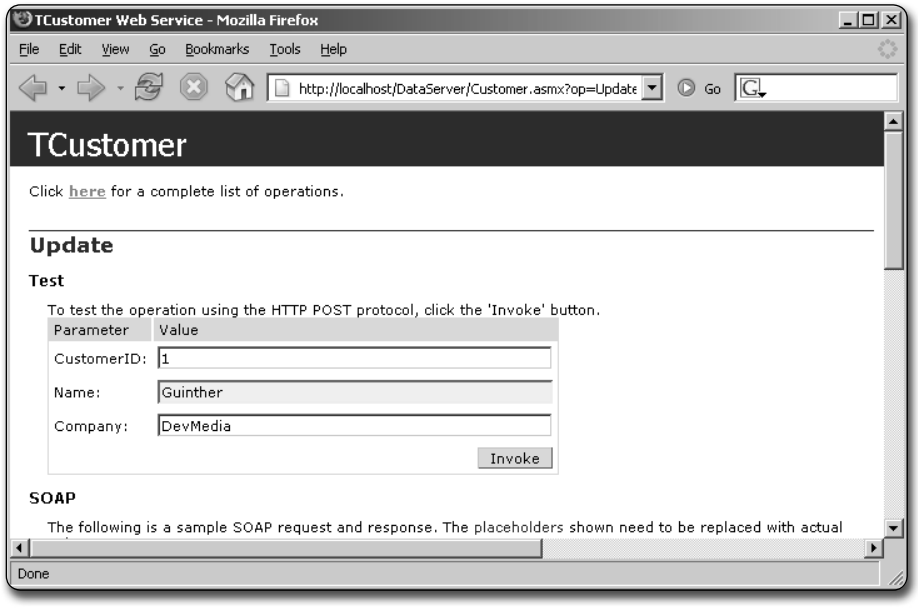


Figura 2. Testando o Web Service

de negócio. Essa é uma das grandes vantagens do desenvolvimento em camadas.

Por trabalhar com SOAP e XML, nosso servidor pode ser agora acessado por uma variedade enorme de linguagens e plataformas: .NET, Win32, Linux, Mobile, Java, PHP, Kylix, VS etc. Veremos alguns exemplos a seguir.

Criando a aplicação cliente ASP.NET

Passamos para a terceira camada da nossa solução, vamos criar uma aplicação Web usando ASP.NET. Inicie um novo projeto ASP.NET no Delphi dando o nome de "ClientWeb".

O primeiro passo para acessar o servidor é adicionar uma *Web Reference*, opção que pode ser acessada a partir de um clique de direita no nome do projeto no *Project Manager*. Na caixa de diálogo que aparece digite o endereço do documento WSDL do servidor, nesse caso:

```
http://localhost/DataServer/Customer.asmx?WSDL
```

Clique na seta, visualize o documento WSDL e na opção *Web reference folder name* digite "DataServer". Clique em *Add Reference*. Note que algumas referências serão adicionadas ao projeto, no *Project Manager*. Um dos procedimentos feitos nesse passo é a criação de uma classe *proxy*, que vai permitir que métodos sejam chamados através de um objeto local, que se encarrega de invocar os métodos remotos via SOAP.

Desenhe uma interface semelhante a mostrada na **Figura 3** (já em tempo de execução). Temos um botão e um *TextBox* ("tbFind), destinados a executar a pesquisa. Um *DataGrid* vai ser usado para exibir os dados retornados.

Logo abaixo, três controles *TextBox* serão usados para permitir a edição dos dados (outra opção seria permitir a edição diretamente no *DataGrid*). Note que o *DataGrid* tem um botão *Select*, para adicioná-lo acesse o *Property Builder* no *Object Inspector*. No editor que aparece selecione *Columns*, depois adicione uma coluna do tipo *Button Column>Select*.

A **Listagem 6** mostra o código do *Find*, nele estamos utilizando a *Web Reference* configurada anteriormente. *TCustomer* parece ser a nossa classe servidora, mas na verdade é a classe *proxy*, que repassa os métodos para o servidor (veja a sua implementação na unit *DataServer.Customer.pas*).

Usando o *SelectDS*, executamos o método remoto e obtemos um *DataSet* de retorno, trafegado em formato XML. A seguir, colocamos esse *DataSet* em sessão e exibimos os dados no *DataGrid*.

Quando o usuário selecionar um registro no *DataGrid*, pressionando o *Select*, precisamos repassar os valores para os *TextBoxes* que estão abaixo. Isso é feito no evento *SelectedIndexChanged* do *DataGrid* (**Listagem 7**). Note que obtemos os valores a partir das células do *DataGrid*, outra opção seria obter a partir do *DataSet*.

O código do *Update* (**Listagem 8**) captura os dados editados nos *TextBoxes* e repassa-os ao *DataSet*. Como o *DataSet* foi alterado, precisamos enviar essas atualizações para o servidor, através do *UpdateDS*.

Não precisamos enviar de volta todo o *DataSet* com todos os registros, somente aqueles que foram alterados. Podemos dizer ao *DataSet* que precisamos somente dos registros alterados, através do método *GetChanges* (pense nisso com um *Delta* do *ClientDataSet*).

O código do *Insert* (**Listagem 9**) é bastante semelhante. Capturamos os valores digitados nos *TextBoxes* e colocamos em um novo registro, representado no ADO.NET pela classe *DataRow* (aqui é o nosso objeto *row*).

Configuramos os campos do registro (*row*) e o adicionamos na coleção de *Rows* do primeiro *DataTable* do *DataSet* (veja que esse procedimento de inserção é bem diferente do modelo *TDataSet* da VCL). O próximo passo é enviar os dados ao servidor através do nosso método remoto *InsertDS*.

A **Figura 3** mostra o exemplo em execução, usando um filtro para seleção dos registros.

Conversão DataSet (ADO.NET) para ClientDataSet

Finalizada a interface Web da solução, vamos passar para a criação da aplicação

Listagem 6. Código do botão Find

```
uses DataServer.Customer;
...
procedure TWebForm1.btnFind_Click(
  sender: System.Object; e: System.EventArgs);
var
  ws: DataServer.Customer.TCustomer;
  ds: DataSet;
begin
  ws := DataServer.Customer.TCustomer.Create();
  ds := ws.SelectDS(TextBox1.Text);
  Session['ds'] := ds;
  DataGrid1.DataSource := ds;
  DataGrid1.DataBind();
end;
```

Listagem 7. Evento SelectedIndexChanged

```
procedure TWebForm1.DataGrid1_SelectedIndexChanged(
  sender: System.Object; e: System.EventArgs);
begin
  tbCustomerID.Text :=
    DataGrid1.SelectedItem.Cells[1].Text;
  tbFirstName.Text :=
    DataGrid1.SelectedItem.Cells[2].Text;
  tbCompany.Text :=
    DataGrid1.SelectedItem.Cells[3].Text;
end;
```

Listagem 8. Código do botão Update

```
procedure TWebForm1.btUpdate_Click(
   sender: System.Object; e: System.EventArgs);
var
  ws: DataServer.Customer.TCustomer;
  ds: DataSet;
  i: integer;
begin
  ds := Session['ds'] as DataSet;
  i := DataGrid1.SelectedIndex;
  ds.Tables[0].Rows[i]['FirstName'] :=
    tbFirstName.Text;
  ds.Tables[0].Rows[i]['Company'] := tbCompany.Text;
  ws := DataServer.Customer.TCustomer.Create();
  ds := ds.GetChanges();
  ws.UpdateDS(ds);
  DataGrid1.DataSource := Session['ds'] as DataSet;
  DataGrid1.DataBind();
end;
```

cliente desktop, em Delphi 7 Win32. Aqui temos um grande problema, relativo à interoperabilidade ADO.NET e *ClientDataSet*.

Como dito anteriormente, nossa camada servidora pode ser acessada por qualquer plataforma. Porém, o método de seleção devolve *DataSets* em XML no formato do ADO.NET. E o *ClientDataSet* do Delphi 7 não consegue carregar um XML nesse formato. Bom, temos várias possíveis soluções para o problema.

Poderíamos na aplicação cliente transformar o XML do ADO.NET em um *DataPacket*, usando o utilitário XML Mapper e componentes de transformação da paleta *Data Access* (veja meu artigo da edição 36). Poderíamos também usar um *XmlDocument* para ler o documento e fazer o mapeamento no braço para o *ClientDataSet*. Ou ainda deixar de lado o *ClientDataSet* e usar controles não data-aware para trabalhar no XML.

Dentre as várias opções, a minha escolha foi a mais ousada e livra o cliente desse processamento extra (que é um dos objetivos do desenvolvimento multicamadas). Eu fiz o próprio servidor devolver um formato aceito pelo *ClientDataSet*, nesse caso o *DataPacket*.

Esse procedimento é trabalhoso e envolve um profundo conhecimento do formato usado pelo DataSnap. Para um bom entendimento das explanações e do código a seguir, sugiro que o leitor observe com cuidado o documento *DataPacket* em XML mostrado na **Figura 4**.

Note que o *DataPacket* tem duas seções principais, *MetaData* e *RowData*. Uma define os campos e tipos, e a outra contém os registros com os dados. É exatamente esse *packet* que precisamos montar "no braço", com base no *DataSet*.

Vamos então precisar modificar o servidor (abra o projeto novamente) para incluir um novo método, responsável por devolver os dados em formato *XML DataPacket*. Chamei o método de *SelectDP* (**Listagem 10**). O sufixo "DP" no nome do método vem obviamente de "DataPacket".

Ele não faz muita coisa, na verdade chama o método *SelectDS* (que retorna um *DataSet* ADO.NET) e "passa a bola" para uma classe que criei, vista a seguir.

Como temos um *DataSet* do ADO.NET e precisamos retornar um *XML DataPacket* a ser reconhecido pelo *ClientDataSet*, preci-

Listagem 9. Código do botão Insert

```
procedure TWebForm1.btInsert_Click(
    sender: System.Object; e: System.EventArgs);
var
  ws: DataServer.Customer.TCustomer;
  ds: DataSet;
  row: DataRow;
begin
  ds := Session['ds'] as DataSet;
  row := ds.Tables[0].NewRow();
  row['FirstName'] := tbFirstName.Text;
  row['Company'] := tbCompany.Text;
  ds.Tables[0].Rows.Add(row);
  ws := DataServer.Customer.TCustomer.Create();
  ds := Session['ds'] as DataSet;
  ds := ds.GetChanges();
  ws.InsertDS(ds);
  DataGrid1.DataSource := Session['ds'] as DataSet;
  DataGrid1.DataBind();
end;
```

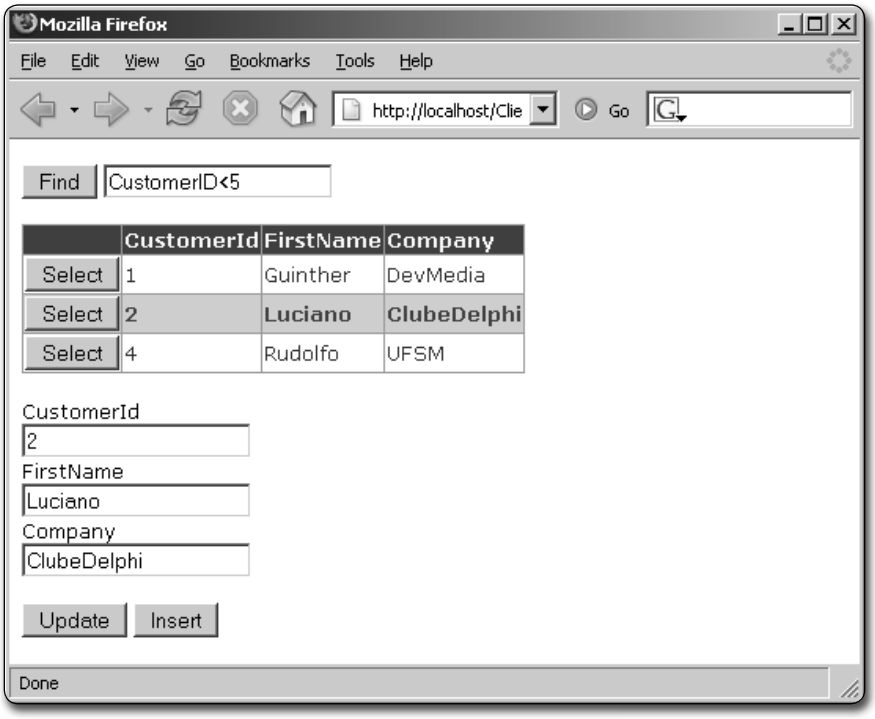
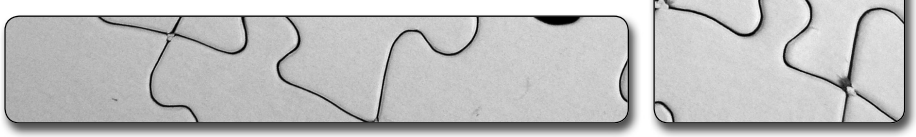


Figura 3. Cliente ASP.NET fornece uma interface Web

```
- <DATAPACKET Version="2.0">
  - <METADATA>
    - <FIELDS>
        <FIELD attrname="CustomerId" fieldtype="i4"/>
        <FIELD attrname="FirstName" fieldtype="string" WIDTH="255"/>
        <FIELD attrname="Company" fieldtype="string" WIDTH="255"/>
      </FIELDS>
      <PARAMS DEFAULT_ORDER="1" PRIMARY_KEY="1"/>
    </METADATA>
  - <ROWDATA>
      <ROW CustomerId="1" FirstName="Guinther" Company="DevMedia"/>
      <ROW CustomerId="2" FirstName="Luciano" Company="ClubeDelphi"/>
    </ROWDATA>
  </DATAPACKET>
```

Figura 4. Gerando um DataPacket no "braço"



samos fazer uma conversão. Resolvi então criar uma classe que faz esse mapeamento, capturando cada definição de campo e registro do *DataTable* do *DataSet* e gerando um XML no formato *DataPacket*.

A classe foi chamada, apropriadamente, de "AdoMidasConverter", na unit *AdoMidas* (**Listagem 11**).

É muito fácil gerar documentos XML no .NET Framework. A classe *XmlTextWriter* possui métodos bastante intuitivos para gerar o documento desejado. A primeira parte do código gera a seção *Metadata*. Usando o novo comando *for in* do compilador percorremos todas as colunas do *DataTable* e vamos adicionando aos *Fields* do *DataPacket*.

Durante a transformação foi necessário fazer um mapeamento do tipo do campo. Por exemplo, o tipo *System.String* do .NET foi mapeado para *string*. O *System.Int32* (de 32 bits) foi mapeado para "i4" (inteiro de 4 bytes). Converti aqui somente os campos utilizados neste exemplo, fique a vontade para mapear os demais tipos de dados que desejar, como datas e *floats*.

A próxima seção é a *RowData*. Diferente do XML do ADO.NET, que representa cada valor de um campo como um elemento no documento XML, o *DataPacket* do Delphi representa valores de campos como atributos. Essa foi uma boa idéia da equipe que projetou o MIDAS, pois ajuda a reduzir o tamanho do XML e facilita o tráfego no HTTP.

Os dados do documento gerado estão guardados em um *MemoryStream*, usado pelo *XmlTextWriter,* como especificamos no construtor. O último passo consiste em obter esses dados e jogar em um *XmlDocument*, que é a classe do .NET utilizada para representar documentos XML.

Já que o *XmlDocument* é serializável, o devolvemos como valor de retorno do *SelectDP,* que está pronto para receber chamadas SOAP vindas do *ClientDataSet* (lembre-se de dar um *Build* no servidor antes de prosseguir).

Criando a aplicação cliente Delphi 7 (Win32)

Vamos criar a segunda interface cliente para a nossa solução, uma aplicação Desktop escrita em Delphi 7. Incie uma nova aplicação VCL e salve todos os arquivos. Dê ao projeto o nome de "ClientWin32".

Listagem 10. Método para devolver DataPackets

```
uses System.Xml ...
public
  [WebMethod]
  function SelectDP(Filter: string): XmlDocument;
...
uses AdoMidas; //criada a seguir
...
function TCustomer.SelectDP(
  Filter: string): XmlDocument;
var
  Dst: DataSet;
begin
  Dst := SelectDS(Filter);
  Result := AdoMidasConverter.DataSetToMidas(Dst);
end;
```

Listagem 11. Classe para converter DataSets do ADO.NET em XML DataPackets do ClientDataSet

```
// Classe para interoperabilidade ADO.NET e MIDAS - Guinther Pauli
unit AdoMidas;
interface
uses
  System.Data, System.Xml, System.IO, System.Text;
type
  AdoMidasConverter = class
  public
    class function DataSetToMidas(
      Dst: DataSet): XmlDocument;
  end;

implementation
{ AdoToMidasConverter }
class function AdoMidasConverter.DataSetToMidas(
  Dst: DataSet): XmlDocument;
var
  writer: XmlTextWriter;
  Mem: MemoryStream;
  Col: DataColumn;
  ColType: string;
  ColName: string;
  Row: DataRow;
  Xml: XmlDocument;
  st: string;
begin
  Mem := MemoryStream.Create();
  writer := XmlTextWriter.Create(Mem, System.Text.ASCIIEncoding.Create);
  writer.Formatting := Formatting.Indented;
  writer.Indentation := 4;
  writer.WriteStartDocument(True);
  writer.WriteStartElement('DATAPACKET');
  writer.WriteAttributeString('Version','2.0');
  writer.WriteStartElement('METADATA');
  writer.WriteStartElement('FIELDS');
  for Col in Dst.Tables[0].Columns do
  begin
    writer.WriteStartElement('FIELD');
    writer.WriteAttributeString('attrname', Col.ColumnName);
    if Col.DataType.ToString() = 'System.String' then
       ColType := 'string';
    if Col.DataType.ToString() = 'System.Int32' then
       ColType := 'i4';
    writer.WriteAttributeString('fieldtype', ColType);
    if ColType = 'string' then
      writer.WriteAttributeString('WIDTH', '255');
    { mapear demais campos aqui }
    writer.WriteEndElement;
  end;
  writer.WriteEndElement; { fields }
  writer.WriteStartElement('PARAMS');
  writer.WriteAttributeString('DEFAULT_ORDER', '1');
  writer.WriteAttributeString('PRIMARY_KEY', '1');
  writer.WriteEndElement; { params }
  writer.WriteEndElement; { metadata }
  writer.WriteStartElement('ROWDATA');
  for Row in Dst.Tables[0].Rows do
  begin
    writer.WriteStartElement('ROW');
    for Col in Dst.Tables[0].Columns do
    begin
      ColName := Col.ColumnName;
      writer.WriteAttributeString(ColName, Row[ColName].ToString());
    end;
    writer.WriteEndElement;
  end;
  writer.WriteEndElement;
  writer.WriteEndDocument();
  writer.Close;
  st := Encoding.UTF8.GetString(Mem.ToArray);
  Xml := XmlDocument.Create();
  Xml.LoadXml(st);
  Result := Xml;
end;
end.
```

O primeiro passo é fazer referência ao servidor, semelhante ao que fizemos no Delphi for .NET através da opção *Add Web Reference*. No Delphi 7, essa opção está em *File>New>Other>WebServices>WSDL Importer*. Informe o caminho do documento WSDL do nosso servidor, para este exemplo é:

```
http://localhost/DataServer/Customer.asmx?WSDL
```

Clique em *Next* e *Finish*. Salve o novo arquivo criado como "Customer.pas". O código gerado automaticamente pelo Delphi é como uma "classe proxy", que fornece os mecanismos necessários para invocar os métodos remotos através do SOAP.

Agora desenhe uma interface semelhante à mostrada na **Figura 5** (já em execução). Nela temos dois botões (*Find* e *Update*), um *Edit* para que seja informado o parâmetro ("EdtFind") e um *DBGrid*. Colocamos também um *ClientDataSet* ("cds") e um *DataSource*. Relacione os componentes apropriadamente.

Para acessar o servidor SOAP, usaríamos normalmente um HTTPRIO da paleta *WebServices*. A boa nova é que a unit gerada pelo Delphi já contém uma função pronta que instancia e configura esse objeto para nós, chamada aqui de *GetTCustomerSoap*.

Na **Listagem 12** temos o código do *Find*, que faz uso dessa função. Nele chamamos o *SelectDP*, que no servidor vai fazer o *select* no banco de dados, obter um *DataSet* do ADO.NET, converter em *XML DataPacket* (através do nosso "engine" *AdoMidas*) e devolver para a aplicação cliente.

Para carregar o XML de retorno no *ClientDataSet*, não é necessário usar o *LoadFromFile*, o que nos obrigaria a salvar o documento no disco. Usamos um *TStringStream* para obter o conteúdo de retorno SOAP (o *DataPacket*) e a seguir o método *LoadFromStream* do *ClientDataSet* é chamado para carregar os dados diretamente da memória. Execute a aplicação e faça alguns testes.

Para atualizar os dados obviamente não adianta chamar o *ApplyUpdates* do *ClientDataSet*. Temos aqui novamente várias possíveis soluções, como criar um método no servidor que recebesse o *delta* e transformasse de volta em um *DataSet* do ADO.NET. Outra solução seria usar os métodos padrão já definidos no servidor, como *update* e *insert*, e apenas invocar esses métodos.

Foi o que fiz na **Listagem 13.** Foi necessário examinar o *UpdateStatus* de cada registro do *ClientDataSet*. Se for encontrada alguma modificação, o método apropriado é invocado no servidor, atualizando corretamente os dados no BD.

E com isso temos uma aplicação cliente desktop 100% funcional, escrita em D7, acessando uma camada intermediária em .NET, comunicando-se através de

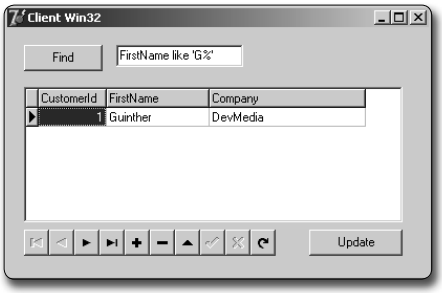


Figura 5. Cliente em Delphi 7 Win32

Listagem 12. Obtendo dados em DataPacket do servidor

```
uses Customer;
...
procedure TForm1.BtnFindClick(Sender: TObject);
var
  Xml: string;
  Mem: TStream;
begin
  Xml := GetTCustomerSoap().
         SelectDP(EdtFind.Text);
  Mem := TStringStream.Create(Xml);
  cds.LoadFromStream(Mem);
   Mem.Free();
end;
```

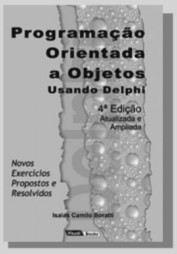
Listagem 13. Atualizando os dados

```
procedure TForm1.BtnApplyClick(
          Sender: TObject);
begin
  cds.First;
  while not cds.Eof do
  begin
    if cds.UpdateStatus = usModified
then
  begin
    GetTCustomerSoap().Update(
     cds.FieldByName('CustomerId').
     AsInteger,cds.FieldByName('FirstName').
     AsString,cds.FieldByName('Company').
     AsString);
    end;
    if cds.UpdateStatus = usInserted
then
  begin
    GetTCustomerSoap().Insert(
     cds.FieldByName('FirstName').
     AsString,cds.FieldByName('Company').
     AsString);
    end;
    // trate a exclusão aqui
    cds.Next;
  end;
end;
```


Lançamentos!

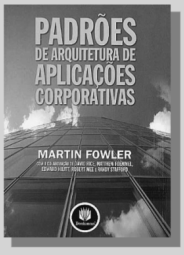
Implementando um Escritório de Projetos
Brasport
R$ 41,00

Programação Orientada a Objetos
Usando Delphi
4ª edição
Visual Books
R$ 75,00



SQL Guia Prático
2ª edição
Brasport
R$ 53,00

Padrões de Arquitetura de Aplicações Corporativas
Bookman
R$ 112,00



COMPRE JÁ e garanta um DESCONTO ESPECIAL!
Utilize o código de cupom DEL06 quando solicitado.*
* Válido até 31/07/2007

Frete Grátis
Para todo o Brasil!

Até 6x sem juros no cartão.*
*Parcela mínima de R$ 20,00

Outras facilidades de pagamento:

Tel: (11) 4062-5152

livrosdeprogramacao.com.br

SOAP. Uma combinação perfeita que pode garantir o sucesso de seus projetos Delphi.

Criando a aplicação Mobile (Pocket PC)

Antes de iniciar qualquer exemplo, gostaria de deixar bem claro aqui que, o desenvolvimento para aplicativos móveis com o Delphi ainda é bem limitado e complicado. Não existem *designers*, nem ferramentas apropriadas para criar aplicativos para Compact Framework.

O que existe são um compilador e bibliotecas, o que torna o desenvolvimento possível, mas se você tiver paciência. Coloquei na seção *Links* várias referências que sugiro a leitura antes de iniciar o desenvolvimento com CF. Ali constam informações sobre configuração, compilação, uso de emuladores etc. Não entrarei em todos esses detalhes aqui.

Nota: Para uma introdução ao desenvolvimento com Delphi e Compact Framework, sugiro ao leitor ver a edição 67.

Nesta última etapa da nossa solução, criaremos então um cliente baseado em aplicativos móveis, mais especificamente o Pocket PC. O aplicativo utilizará Web Services para se comunicar com o servidor, nesse caso, é necessário que o dispositivo tenha acesso a mesma rede onde se encontra o servidor, por exemplo, via Wi-Fi, ou acesso Internet que permite conectar o servidor WS. Uma alternativa ao modelo conectado seria criar uma versão desconectada (gravando os dados em XML local ou usando o SQL Server).

No BDS, inicie um nova aplicação Delphi for .NET do tipo Windows Forms, chamando o projeto de "ClientMobile". Adicione uma *Web Reference* da mesma forma que foi feito para a aplicação ASP.NET. Uma diferença aqui é que você deve colocar o endereço IP do adaptador *Loopback Adapter* ao invés de *localhost* no endereço do servidor.

Nota: Você deve adicionar um *Loopback Adapter* para testar as aplicações com CF. Isso pode ser feito a partir do item *Adicionar/Remover Hardware>Adaptadores de Rede*. A seguir atribua um IP fixo para o mesmo. Mais informações sobre esse procedimento podem ser encontradas na seção *Links*.

Desenhe o formulário mostrado na **Figura 6.** Temos três *TextBoxes* ("tbFind", "tbFirstName" e "tbCompany"), dois botões (*Find* e *Update*) e um *DataGrid*, além de alguns *Labels*.

A **Listagem 14** mostra o código completo do formulário. No evento *Load* do formulário instanciamos uma variável

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a aulas de Jefferson Junglaus que mostra como trabalhar com o Compact Framework no Delphi.

www.devmedia.com.br/articles/listcomp.asp?txtsearch=Compact+Framework+no+Delphi

Listagem 14. Código do formulário para CF

```
uses DataServer.Customer
...
public
  ds: DataSet;
  ws: DataServer.Customer.TCustomer;
...
procedure TWinForm1.TWinForm1_Load(
  sender: System.Object; e: System.EventArgs);
begin
  ws := DataServer.Customer.TCustomer.Create();
end;
procedure TWinForm1.btFind_Click(
   sender: System.Object; e: System.EventArgs);
begin
  ds := ws.SelectDS(tbFind.Text);
  DataGrid1.DataSource := ds.Tables[0];
  DataGrid1_CurrentCellChanged(nil,nil);
end;
procedure TWinForm1.DataGrid1_CurrentCellChanged(sender: System.Object; e: System.EventArgs);
var
  row: DataRow;
begin
  row := ds.Tables[0].Rows[dataGrid1.CurrentRowIndex];
  tbFirstName.Text := row['FirstName'].ToString();
  tbCompany.Text := row['Company'].ToString();
end;
procedure TWinForm1.btUpdate_Click(sender: System.Object; e: System.EventArgs);
var
  row: DataRow;
begin
  row := ds.Tables[0].Rows[dataGrid1.CurrentRowIndex];
  row['FirstName'] := tbFirstName.Text;
  row['Company'] := tbCompany.Text;
  ws.UpdateDS(ds);
end;
```

Listagem 15. Arquivo BAT para compilar aplicações CF

```
set _DCCILPATH_="c:\program files\borland\BDS\4.0\bin\dccil.exe"
set _CFUNITS_="c:\program files\borland\BDS\4.0\lib\cf"
set _PROJECTNAME_=%1
set _PROJECTDIR_=%2
set _BDSDIR_=..\BDS\4.0\Bin
path %_BDSDIR_%;%PATH%
SHIFT
SHIFT
:LOOP
IF "%1" == "" GOTO END
set _PROJECTDIR_=%_PROJECTDIR_% %1
SHIFT
GOTO LOOP
:END
cd %_PROJECTDIR_%
rem del *.dcuil
%_DCCILPATH_% "%_PROJECTDIR_%%_PROJECTNAME_%" -u%_CFUNITS_% -luSystem.Windows.Forms
-luSystem.Data
Pause
```

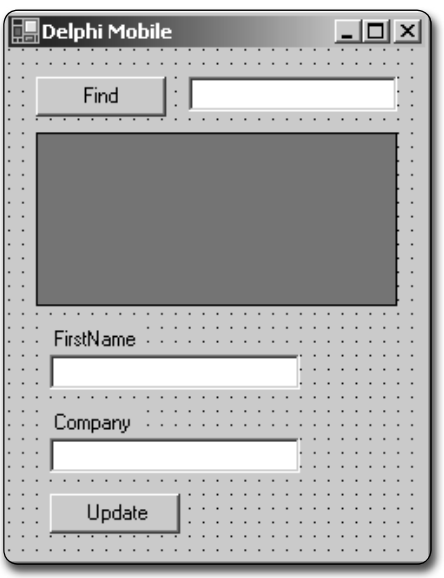


Figura 6. Aplicação Compact Framework

chamada *ws* (declarada no form), que representa uma referência ao Web Service. No botão *Find* chamamos o *SelectDS* do servidor, que devolve um *DataSet* do ADO.NET que é usado como *DataSource* do *DataGrid*.

O evento *CurrentCellChanged* do *DataGrid* verifica que registro foi selecionado e repassa os valores para os *TextBoxes*. O *Update* simplesmente captura os valores dos *TextBoxes* e repassa ao WebService, para atualização.

Para não estender demais o exemplo não codifiquei a inserção e exclusão, é semelhante ao *Update*.

Compilando para Compact Framework

Vamos precisar fazer alguns ajustes na aplicação para que ela possa ser compilada para CF. O primeiro passo é no *Project Manager,* no item *References*, adicionar uma referência ao assembly *System.Windows.Forms.DataGrid.dll* (localize esse assembly no seu disco).

O próximo passo é retirar as seguintes linhas do arquivo do projeto (*Project>View Source*):

```
{$R 'WinForm1.TWinForm1.resources' 'WinForm1.resx'}
...
[STAThread]
```

A seguir, no arquivo do formulário, algumas alterações devem ser feitas no *InitializeComponent* do formulário, que contém todo o código para inicialização dos controles. Essas alterações são:

- Comentar a linha *Self.SuspendLayout*;
- Comentar a linha *Self.AutoScaleBase Size...;*
- Comentar a linha *Self.ResumeLayout(False);*
- Comentar todas as atribuições às propriedades *Name* e *TabIndex* de todos os controles;
- No *DataGrid* manter somente as linhas que configuram o *Location*, *Size* e *Text*, as demais linhas devem ser todas comentadas.

> **Nota**: Uma simples modificação em um controle no *form designer* fará todas essas alterações serem desfeitas, então certifique-se que o mesmo esteja totalmente pronto e configurado antes de aplicar as configurações.

Acesse agora o menu *Tools>Configure Tools*. Adicione um novo item com as configurações mostradas a seguir:

```
Title = Delphi .NET CF Preview Compiler
Program = C:\Program Files\Borland\BDS\4.0\
          Bin\cf.bat
Parametres = $SAVEALL $NAMEONLY($PROJECT).dpr
$PATH($PROJECT)
```

Crie o arquivo BAT da **Listagem 15** dentro do diretório *Bin* do Delphi, certificando-se que os caminhos mostrados sejam o mesmo da sua máquina. O nome do arquivo deve ser "cf.bat".

A partir do menu *Tools*, acesse o novo item criado e uma janela de *prompt* deve aparecer indicando o processo de compilação da aplicação CF. Se algo der errado, observe a mensagem de erro do compilador e verifique o seu código-fonte.

O último passo é fazer o *deploy* para um emulador (caso não tenha um Pocket PC). Existem vários emuladores que você pode utilizar para testar a aplicação criada (para download de emuladores, veja a seção *Links*).

Nesse exemplo foi utilizado o Pocket PC 2002 Emulator. A aplicação Delphi rodando no Pocket PC, acessando nosso Web Service, aparece na **Figura 7**.

No meu caso, para fazer o *deploy* da aplicação para o Pocket PC Emulator 2002, compartilhei primeiramente o diretório onde compilei a aplicação. Depois simplesmente iniciei o dispositivo e no *File Explorer* fiz o mapeamento do diretório do servidor usando a seguinte sintaxe "\\NomeDaMaquina\\DiretorioCompartilhado". Copiado o EXE, foi só executar e conferir o resultado.

Comprove a flexibilidade da nossa solução executando os três aplicativos cliente simultaneamente, vendo que a alteração nos dados feita em um cliente é automaticamente refletida no outro.

Conclusão

O desenvolvimento de aplicações em camadas com SOAP, XML e Web Services, oferece inúmeros benefícios: independência de plataforma, linguagem e dispositivo, acesso remoto de uma variedade de ambientes através da Internet, separação do acesso a dados / lógica de negócio das aplicações clientes, que cuidam apenas da UI, entre outras.

A aplicação apresentada neste artigo vai permitir a você oferecer ao seu cliente uma solução bastante robusta, flexível e inovadora, desfrutando de todos os

Figura 7. Aplicação Delphi rodando no Pocket acessando o WS e BD

recursos oferecidos pelo ambiente Web (como distribuição) aliados às facilidades do desenvolvimento RAD e desktop do Delphi. Tudo combinado em harmonia em uma única arquitetura, que conversa a mesma linguagem: XML.

Na próxima edição vamos continuar explorando o universo do desenvolvimento multicamadas, onde veremos como construir uma camada cliente Web desenvolvida com ASP.NET que faz acesso a um servidor de aplicação DataSnap / COM+.

Sucesso nos projetos com XML Web Services, encontro vocês na próxima edição. Grande abraço! ●

Links

SQL Server 2005 Express
www.microsoft.com/sql/editions/express

Artigo sobre Compact Framework com Delphi (plug-ins, emuladores)
dn.codegear.com/article/33507

Usando emuladores
www.nsbasic.com/ce/info/technotes/TN23.htm

Loopback Adapter
www.microsoft.com/technet/prodtechnol/virtualserver/2005/proddocs/vs_operate_using_loopback.mspx?mfr=true

Dicas de DataGrid

Luciano Pimenta
(lucianopimenta@clubedelphi.net)
é Técnico em Processamento de Dados, Editor Técnico da Revista ClubeDelphi, .net Magazine e WebMobile. Editor de vídeo aulas do Portal DevMedia (www.devmedia.com.br). Palestrante da 4ª edição da Borland Conference (BorCon).

Desde a minha primeira experiência com o ASP.NET, um dos controles que mais utilizei foi o *DataGrid*, pela sua facilidade em mostrar os dados no formato de tabela. Adicionar imagens, controles e formatações de acordo com determinado campo, são as primeiras funcionalidades que desejamos nesse controle. O *DataGrid* é muito customizável e neste artigo, veremos algumas dicas úteis para a utilização do mesmo em suas aplicações Web.

Adicionando controles

A funcionalidade de *templates* do ASP.NET ajuda em muito a customização do *DataGrid* para a adição de controles. Basta criar um coluna do controle do tipo *Template Column* e adicionar os controles necessários.

Neste exemplo, vamos trabalhar o *CheckBox*. Crie uma nova aplicação Web no Delphi 2006. Na *WebForm1.aspx* adicione um *DataGrid*. Não vamos no concentrar em conexões com o banco ou formatações do controle. Temos ótimos artigos publicados (edição 66), mostrando essas características, bem como temos vídeos aulas no Portal do Assinante. Apenas indicaremos quando usar *FbCommand* ou *DataSet*.

Formate o *DataGrid* na opção que deseja e conecte ao banco de dados *Employee* que acompanha o Firebird. Você pode usar outro banco de sua preferência. Para este exemplo, use os dados da tabela *Customer* (selecione apenas alguns campos).

Acesse a propriedade *Columns* do *DataGrid*, adicione um campo do tipo *Template Column* e clique em OK (**Figura 1**).

Clique com o botão direito no *DataGrid* e acesse a opção *Columns[0]*. Na seção *ItemTemplate*, adicione um *CheckBox*. Feche o editor, rode a aplicação e veja os *CheckBoxes* adicionados ao *DataGrid*.

Para saber as linhas que foram "selecionadas" com o *CheckBox*, adicione um botão e um *TextBox* na página. No *Click* do

botão adicione o código da **Listagem 1**.

O código está comentando para um fácil entendimento. O que fizemos foi criar uma variável do tipo *StringBuilder* (já adicionando um texto inicial), percorrer todas as linhas do *DataGrid* e procurar o controle *CheckBox* com o *FindControl*.

Esse método retorna um *Control*, então fizemos um *typecast* para *CheckBox*. Vale salientar que o parâmetro aceito pelo *FindControl* é o nome do controle. Se o controle não for nulo (foi encontrado) e estiver marcado (*Checked*), então adicionamos na *sb* os valores das colunas 1 e 2.

> **Nota:** As colunas do *DataGrid* começam com 0, sendo que a do *Template Column* é contada também.

Rode a aplicação, marque alguns *CheckBoxes* e clique no botão. Como mostra a **Figura 2**, o *TextBox* exibe as linhas "selecionadas".

Master/Detail

Criar relacionamentos *master/detail* no *DataGrid* é muito simples. Crie uma nova página e adicione dois *DataGrids*. Para o primeiro, criei uma conexão com a tabela *Department*. Para o segundo acessei os dados da tabela *Employee* com uma cláusula *where* no campo DEPT_NO. No *Page_Load* estou carregando os dados do *DataGrid* mestre, conforme o código da **Listagem 2**.

Código simples para quem está familiarizado com ASP.NET, onde abrimos a conexão com o banco, repassamos para o *DataSource* do *DataGrid* o *DataReader* e chamamos o *DataBind* do controle. Por fim, fechamos a conexão com o banco.

A propriedade *DataKeyField* é muito importante, pois ela dirá o campo chave que estamos selecionando. Adicione no *DataGrid* master uma coluna do tipo *Select*. No evento *ItemCommand* filtraremos o segundo *DataGrid* com o código da **Listagem 3**.

No evento estamos passando para o parâmetro do *FbCommand* a chave da linha selecionada do *DataGrid* master. O restante é igual ao exemplo anterior. Rode a aplicação e selecione um departamento para visualizar os funcionários do mesmo (**Figura 3**).

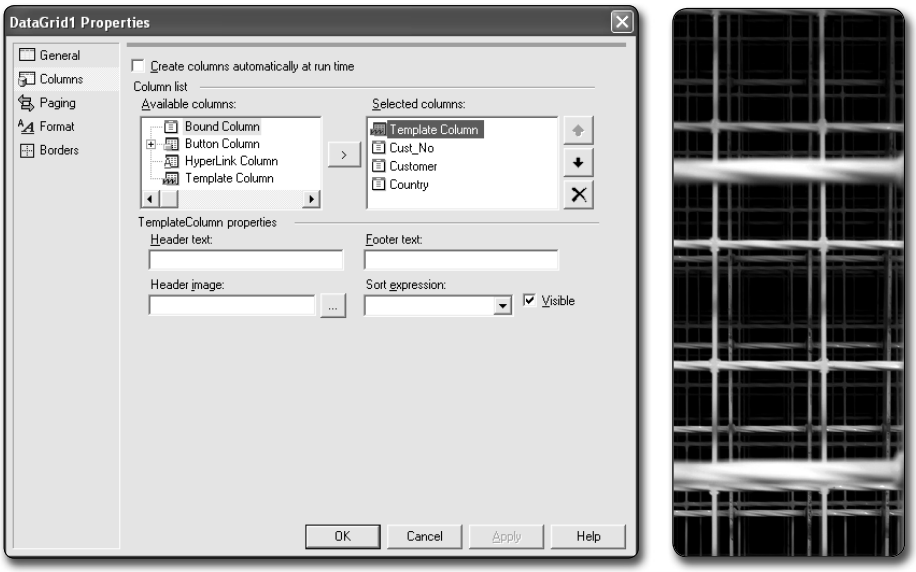


Figura 1. Adicionando um Template Column no DataGrid

Listagem 1. Obtendo a linha com o CheckBox marcado

```
uses System.Text;
...
procedure TWebForm1.Button1_Click(sender: System.Object; e: System.EventArgs);
var
  sb: StringBuilder;
  row: DataGridItem;
  ch: Control;
begin
  { Cria um StringBuilder }
  sb := StringBuilder.Create('* Itens Selecionados *');

  { Percorre todos os itens do DataGrid }
  for row in DataGrid1.Items do
  begin
    { procura pelo CheckBox }
    ch := row.FindControl('CheckBox1');

    { Se achou e esta marcado }
    if (ch <> nil) and (ch as CheckBox).Checked then
    begin
      { Adiciona na var sb os campos do DataGrid }
      sb.Append(#13);
      sb.Append(row.Cells[1].Text + ' - ' + row.Cells[2].Text);
    end;
  end;
  TextBox1.Text := '';
  TextBox1.Text := sb.ToString;
end;
```

Listagem 2. Mostrando os dados do DataGrid master

```
if not IsPostBack then
begin
  FbConnection1.Open;
  try
    DataGrid1.DataSource := FbMaster.ExecuteReader;
    DataGrid1.DataKeyField := 'DEPT_NO';
    DataGrid1.DataBind;
  finally
    FbConnection1.Close;
  end;
end;
```

Listagem 3. Filtro para o DataGrid Detail

```
if e.CommandName = 'Select' then
begin
  FbConnection1.Open;
  try
    FbDetail.Parameters.Add('@DEPT_NO',
      FbDbType.Integer, 4, 'DEPT_NO').Value := DataGrid1.DataKeys[e.Item.ItemIndex];
    DataGrid2.DataSource := FbDetail.ExecuteReader;
    DataGrid2.DataBind;
  finally
    FbConnection1.Close;
  end;
end;
```

Listagem 4. Carregando o DataSet em memória

```
if not IsPostBack then
begin
  { Preenche o DataSet }
  FbMaster.Fill(dsMasterDetail, 'Department');
  { Indica a fonte de dados para o DataGrid master }
  DataGrid1.DataSource := dsMasterDetail.Tables['Department'];
  DataGrid1.DataKeyField := 'DEPT_NO';
  DataGrid1.DataBind;
end;
```

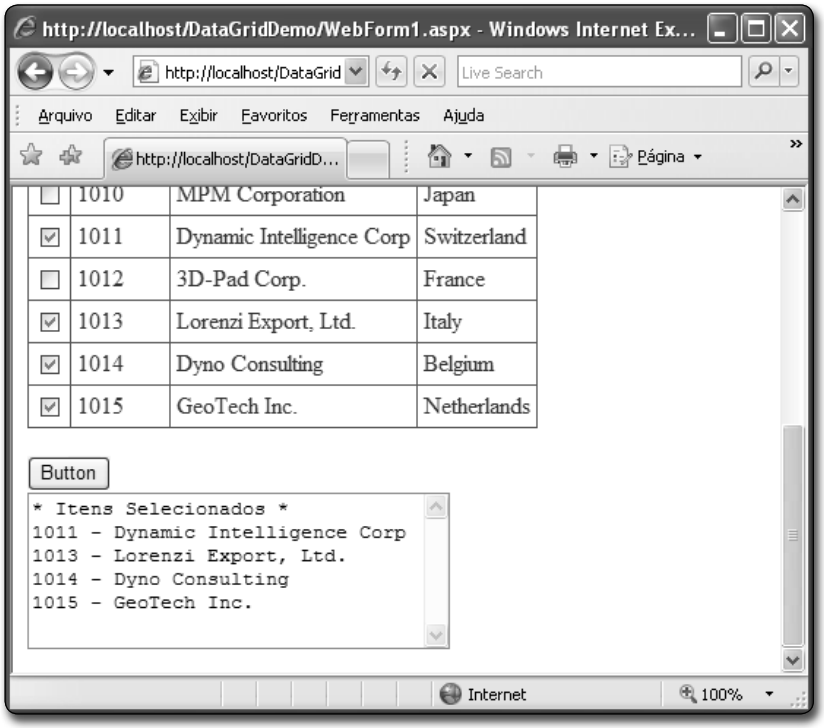


Figura 2. Obtendo os CheckBox marcados

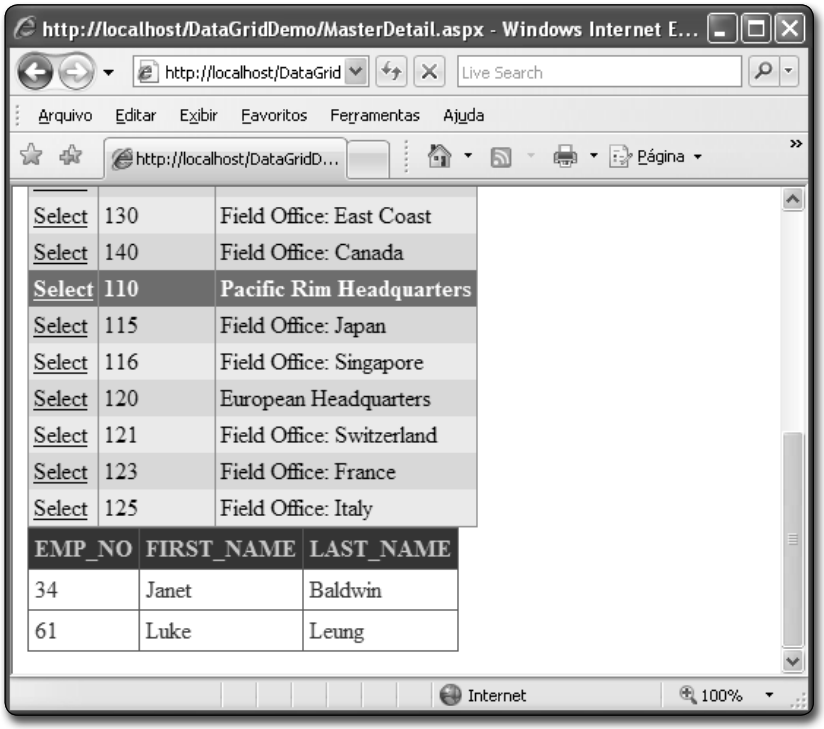


Figura 3. Master/Detail com dois DataGrids

Como você deve ter notado, quando o usuário clica no *DataGrid* master estamos abrindo e fechando uma conexão com o banco. Para poucos dados (como é nosso exemplo) a performance não é afetada, mas imagine uma quantidade maior de dados.

Vamos fazer um exemplo agora usando *DataSet*, pois ele coloca os dados em memória e para filtrar usamos os seus métodos. O código da **Listagem 4** deve ser adicionado no *Page_Load* (crie um novo *Web Form*). Nesse código estamos preenchendo o *DataSet* com uma consulta à tabela *Department*.

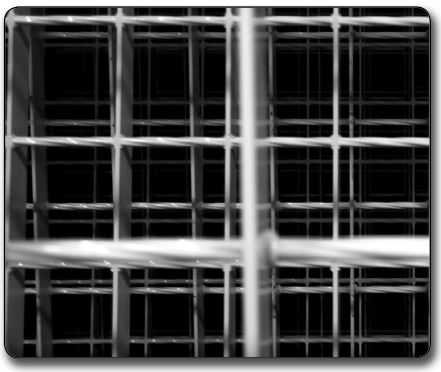
O *dsMasterDetail* é um *DataSet* adicionado no *design* da página. No evento *ItemCommand* do *DataGrid1* colocaremos o código que fará o filtro do *DataTable* detail, usando o código da **Listagem 5**.

Como podemos ver, filtramos os dados do *DataTale* usando o *DefaultView* e repassamos os dados para o *DataGrid2*. Execute a aplicação e veja o resultado (**Figura 4**). Para otimizar o resultado, você pode colocar o *DataSet* em *cache* ou *session*.

Trabalhando com imagens

Para este exemplo, imagine que tenhamos imagens salvas na pasta da aplicação, chamada *imagens*, e na tabela do banco tenhamos apenas o nome dela, ou seja, as imagens não ficam salvas no banco mas no disco, o que é melhor quando se trata de aplicações Web.

O processo para exibir a imagem em um *DataGrid* é simples. Usando o campo que guarda o nome da imagem, você deve abrir o editor de colunas e configurar a propriedade *DataFormatString*, adicionando o seguinte valor: "<img src=imagens/{0}>". Rode a aplicação e veja as imagens sendo mostradas no *DataGrid* (**Figura 5**).



ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta que mostra como fazer um master/detail no DataGrid usando somente um controle DataGrid.

www.devmedia.com.br/articles/viewcomp.asp?comp=2050

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta que mostra como salvar imagens em diretórios e exibi-las no DataGrid.

www.devmedia.com.br/articles/viewcomp.asp?comp=2853

Imagens de acordo com determinada condição

Em uma tabela de clientes, certamente, possuímos um campo indicador do sexo do mesmo. Que tal mostrar no *DataGrid* a listagem dos clientes e de acordo com o sexo mostrar uma imagem diferente?

Criei uma tabela de clientes, onde existe um campo chamado SEXO, salvando as opções "M" ou "F". Vou usar *DataSet* neste exemplo, mas você pode adaptá-lo facilmente para usar *DataReader*. O código da **Listagem 6** vai retornar o tipo da imagem de acordo com o valor do parâmetro.

O código anterior retorna o nome do arquivo, salvo na pasta *imagens* de acordo com o valor passado como parâmetro. No editor de colunas do *DataGrid*, adicione uma do tipo *Template Column*. Acesse o editor de *templates*, adicione um *Image* e feche o mesmo. No evento *ItemDataBound* do *DataGrid* adicione o código da **Listagem 7**.

Com esse código, passamos para propriedade *ImageURL* do controle, o respectivo arquivo de imagem que deve ser mostrado. Rode a aplicação e veja as imagens sendo mostradas de acordo com o sexo do cliente (**Figura 6**).

Se desejar, você pode desmarcar a opção *Visible* no *Property Builder* para o campo *Sexo* não ser mostrado no *DataGrid*, já que existe a figura indicativa.

Somatório no rodapé

Se o *DataGrid* possui um campo com valores, certamente você gostaria de mostrar um total no rodapé da página. E essa técnica é bastante simples. Primeiramente, altere para *True* a propriedade *ShowFooter* do *DataGrid*.

Listagem 5. Filtrando os dados do DataTable

```
if e.CommandName = 'Select' then
begin
  { Preenche o DataSet }
  FbDetail.Fill(dsMasterDetail, 'Employee');
  { Filtra os dados do DataTable com uma expressão }
  dsMasterDetail.Tables['Employee'].DefaultView.RowFilter :=
    'DEPT_NO = '+DataGrid1.DataKeys[e.Item.ItemIndex].ToString;
  { Repassa para o DataGrid Details a fonte de dados }
  DataGrid2.DataSource := dsMasterDetail.Tables['Employee'].DefaultView;
  DataGrid2.DataBind;
end;
```

Listagem 6. Criando um método que retorne o nome do arquivo

```
function TWebForm1.GetImagem(aSexo: string): string;
begin
  if aSexo = 'M' then
    Result := '~/imagens/M.bmp'
  else if aSexo = 'F' then
    Result := '~/imagens/F.bmp'
end;
```

Listagem 7. Indicando a imagem de acordo com o campo SEXO

```
var
  aControl: Control;
begin
  aControl := e.Item.Cells[2].FindControl('Image1');
  if aControl <> nil then
    (aControl as Image).ImageUrl := GetImagem(e.Item.Cells[2].Text);
end;
```

Listagem 8. Colocando um somatório no rodapé

```
if e.Item.ItemType = ListItemType.Footer then
begin
  e.Item.Cells[3].Text := 'Total: '+
   System.&String.Format('{0:c}', DataSet1.Tables[0].Rows[0]['TOTAL']);
end;
```

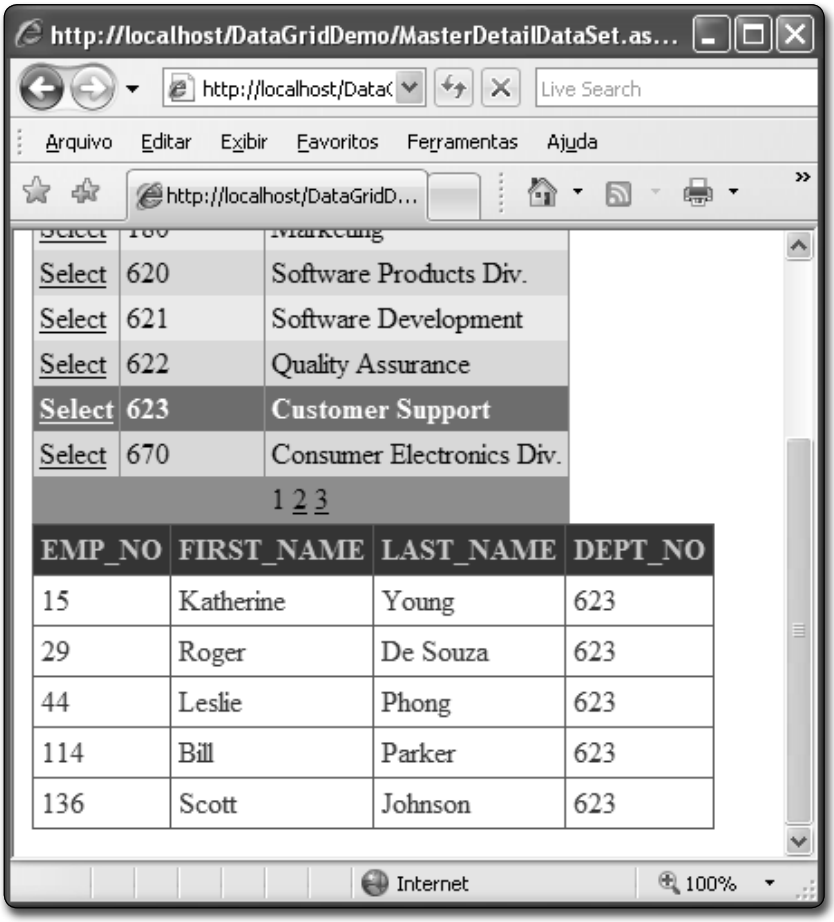


Figura 4. Master/detail usando DataSet e filtro

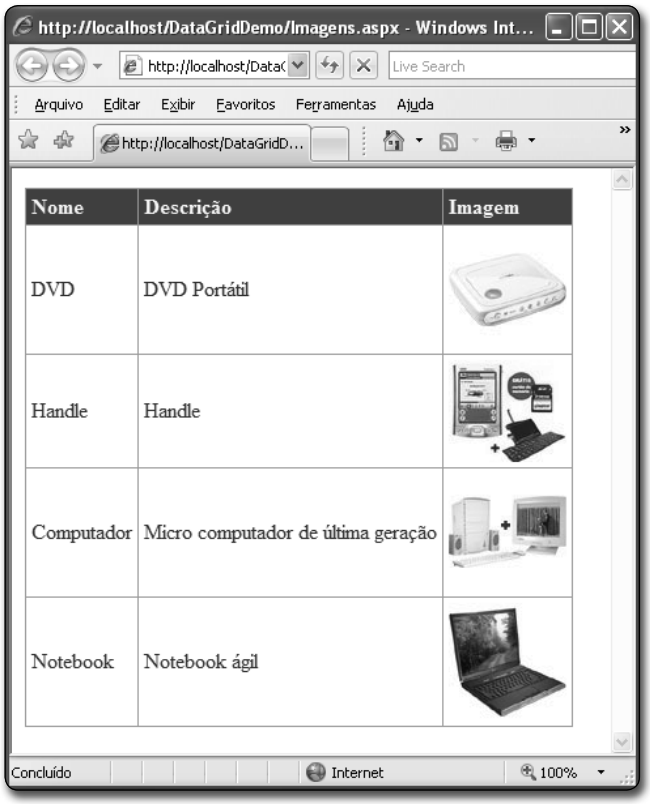


Figura 5. Mostrando imagens da aplicação no DataGrid

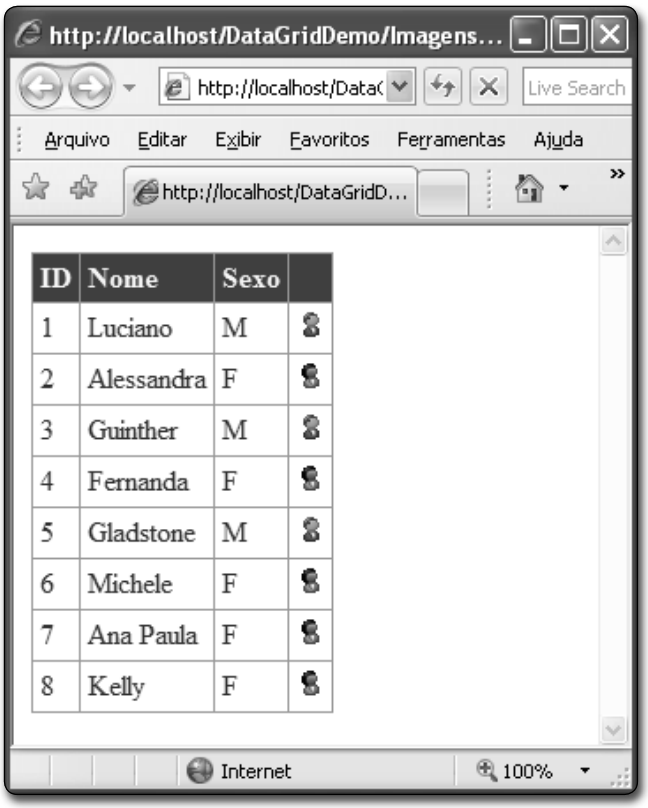


Figura 6. Mostrando imagens de acordo com o valor do campo SEXO

Neste exemplo, vamos utilizar um *DataSet* com um "campo calculado". Após configurar o *DataSet*, adicione uma nova coluna com o nome de TOTAL do tipo *System.Decimal*. No *Page_Load* adicione o seguinte código:

```
FbDataAdapter1.Fill(DataSet1);
DataSet1.Tables[0].Columns['TOTAL'].
Expression := 'SUM(TOTAL_VALUE)';
DataGrid1.DataSource := DataSet1;
DataGrid1.DataBind;
```

Veja que estamos passando para a coluna TOTAL a soma do campo TOTAL_VALUE, usando a propriedade *Expression* da coluna. Acesse o evento *ItemDataBound* e adicione o código da **Listagem 8**.

Verificamos primeiramente se estamos no rodapé do *DataGrid* e depois colocamos na célula o valor total (**Figura 7**).

Conclusão

Vimos neste artigo algumas dicas avançadas para a utilização do *DataGrid*, um dos controles mais utilizados em aplicações ASP.NET.

Para quem quiser avançar, que tal colocar todas essas dicas em prática, estendendo o controle padrão para um *DataGrid* turbinado. Fica aí a dica! Um grande abraço a todos e até a próxima. ●

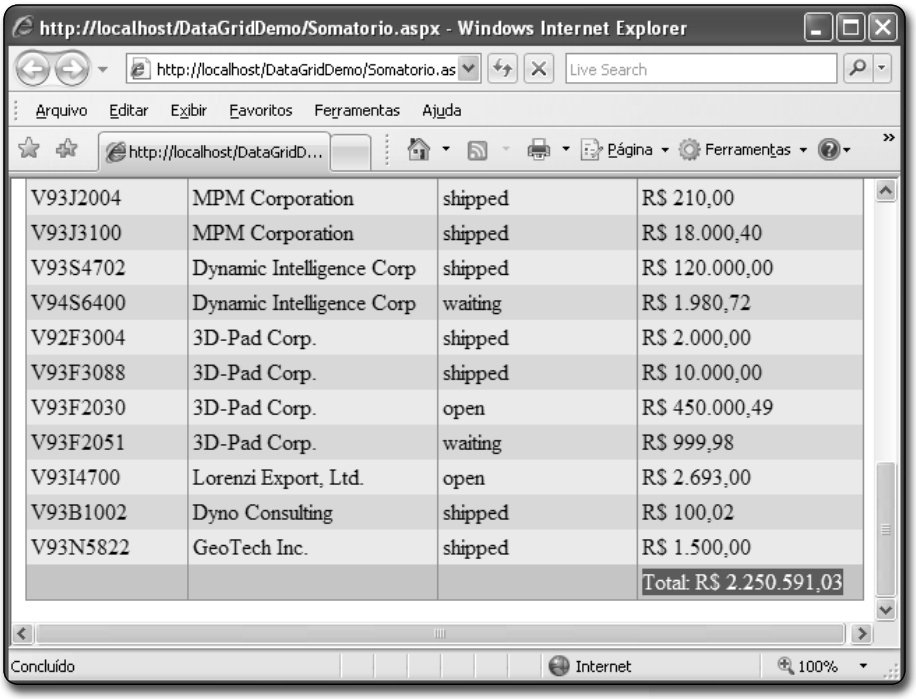


Figura 7. Mostrando o somatório no DataGrid

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta que mostra como criar um somatório no rodapé do DataGrid.

www.devmedia.com.br/articles/viewcomp.asp?comp=2648

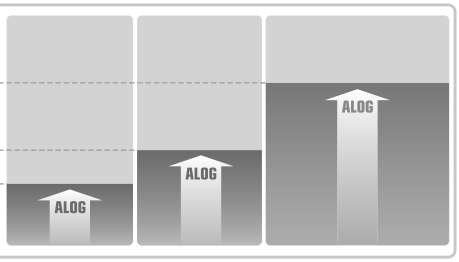
FAÇA COMO A ALOG
FOQUE EM SEU NEGÓCIO

A ALOG é totalmente focada em serviços de Data Center. Dedica-se 24 horas por dia, 7 dias por semana a cuidar do negócio dos nossos 320 clientes corporativos.

ALOG desenvolveu no Brasil o conceito de **HOSTING GERENCIADO**: um serviço de excelência que garantirá a sua empresa estabilidade, agilidade, redundância, e escalabilidade. Para que isso seja verdade os clientes da ALOG contam com **Suporte Incondicional***, um conceito revolucionário em atendimento no Brasil.

ENTENDA O **HOSTING GERENCIADO**

CAMADAS DE NEGÓCIO	DESCRIÇÃO
Processo de Negócio	Atividades cotidianas do cliente
Aplicação do Cliente	Softwares de Gestão (Ex: ERP, CRM, outros)
Ambiente da Aplicação	.NET, JAVA, SQL Server, MySQL, Oracle, Exchange
Sistema Operacional	Windows, Linux, Unix, Solaris
Servidor ou equipamento	Servidores RISC/CISC, Appliances, Roteadores, Switches
Rede	IP, Wan, Rede Local
Infra-estrutura Básica	Energia, HVAC, Segurança - 2N+1



ESTRATÉGIA DO CLIENTE	COLOCATION	HOSTING	HOSTING GERENCIADO

PLANOS DE **HOSTING GERENCIADO**

GERENCIADO 1

- **Pentium D 2.8 GHz**
- 512 RAM DDR2
- 2 HD SATA 80 GB
- 450 GB de Transf. Mensal
- Firewall e Backup

BOOT REMOTO
GERENCIAMENTO**

R$**1.050,00** mensais
INSTALAÇÃO IMEDIATA***

GERENCIADO 2

- **Dual Xeon Dual Core 3.0 GHz**
- 1GB RAM DDR2 533
- 2 HD SATA RAID I 300 GB
- 1.000 GB de Transf. Mensal
- Firewall e Backup

BOOT REMOTO
GERENCIAMENTO**

R$**2.850,00** mensais
INSTALAÇÃO IMEDIATA***

* Sujeito a condições contratuais ** Podendo escolher: HD, RD e GT *** Sujeito a disponibilidade no estoque

ALOG data centers

Excelência em Hosting
www.alog.com.br :: 0800 282 3330

Hosting Gerenciado I Colocation I Contingência I Email Corporativo I Conectividade I Serviços Profissionais

Delphi 2007 em Detalhes

Adriano Santos
(artes@doiscliques.com)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É colunista e membro da Comissão Editorial da revista ClubeDelphi.

Durante todo o mês de Março até o início de Abril, a CodeGear, nova divisão da Borland criada para atender 100% dos desenvolvedores e responsável pelo desenvolvimento do Delphi, JBuilder, C++ Builder e outras ferramentas da empresa, esteve mostrando as novidades para o ano.

Entre elas estão o Delphi for PHP, ferramenta para desenvolvimento de aplicações com PHP, e a nova versão do Delphi para Win32, o Delphi 2007. Estive no evento e pude ver, testar e comprovar de perto os novos recursos da ferramenta.

Neste artigo veremos as principais novidades e faremos alguns exemplos pra demonstrar os novos recursos disponíveis.

Nota: Para uma introdução ao Delphi 2007, consulte a edição 84.

Favoritos

Logo que iniciamos a ferramenta notamos uma leve semelhança com o BDS 2006, porém a página inicial, chamada de *Home* pela CodeGear, contém um item bastante prático: os favoritos. Nesse item é possível adicionar os projetos que são mais utilizados no dia a dia e até criar grupos para o acesso rápido.

Após a abertura ou criação de algum projeto é gravada uma referência do mesmo na seção *Recently Opened Projects* e ao lado de cada projeto aparece o link (*Make me a Favorite*). Basta clicar nesse link que o mesmo será adicionado a seção *My Favorites*.

Nessa última encontramos as opções *Show Favorites* e *Manage Favorites*. Clicando em *Manage Favorites* observamos que ao centro da página inicial é mostrada uma nova seção: *Create New Favorites Group* e nela a opção *Group Title*. Para criar um novo grupo basta digitar o nome dele e clicar em *Create*.

Instantaneamente seu novo grupo aparecerá na seção *Manage Your Favorite Projects* logo acima. Desse momento em diante você poderá mover o atalho para

seu projeto apenas clicando em *Move* e escolhendo o grupo que deverá ser incluído.

Ainda falando de favoritos, a seção Recent Projects, à esquerda da página inicial, contém os últimos cinco projetos abertos e ainda as opções Show Recent Projects, Open Project, New Project e Help, agilizando ainda mais a visualização de últimos projetos, abertura, criação e ainda acesso à ajuda do IDE (**Figura 1**).

MSBuild, Vista e XP Themes

E falando em facilidades ao desenvolvedor, a CodeGear tratou de criar um excelente recurso que auxilia e ajuda a automatizar tarefas na hora do Build. Chamado de *MsBuild*, o recurso dá a oportunidade ao desenvolvedor de determinar tarefas pré (*Pre-build*) e pós (*Post-build*) compilação. Vamos fazer um exemplo simples para entendermos melhor como funciona.

Crie uma nova aplicação no Delphi usando o menu *File>New>VCL Forms Application – Delphi for Win32* e logo em seguida clique em *Project>Options* ou pressione *CTRL+SHIFT+F11*. Com as opções de projeto abertas clique em *Build Events*.

Nesse item encontramos dois grupos de opções: *Pre-Build e Post-Build*. Cada um deles possui um campo para edição e um botão. No primeiro deles digite "calc.exe" e no segundo "notepad.exe". Clique em OK e efetue a compilação do projeto pressionando CTRL+F9.

Nesse ponto notará que a calculadora do Windows se abre, isso porque determinamos que a mesma seria aberta antes da compilação. Para continuar o processo feche a calculadora e note que logo que o IDE encerra o *Build* o Bloco de Notas é mostrado.

Isso facilita a automatização de tarefas, já que no lugar da calculadora e do *NotePad* podemos incluir comandos que serão executados antes e depois da compilação.

> **Nota:** Observe que tanto no pré quanto no pós build o Delphi 2007 fica aguardando até que a tarefa seja efetuada.

Volte ao *Project>Options* e clique agora no botão *Edit* da opção *Pre-Build*. Note que a tela que se abre contém uma série de comandos pré-configurados que podem ser utilizados e mesclados entre si para criação de pequenas tarefas em lote.

Para as opções *Compiler*, *Compiler Messages*, *Linker* e *Directories/Conditionals* foi criada a opção de guardar configurações específicas para cada *Build* da aplicação. Dessa forma você pode organizar melhor as opções do projeto e determinar como cada compilação será efetuada.

Entre em *Project>Options* novamente e clique em uma das opções mencionadas. Note que à direita surge um *ComboBox* chamado *Build Configuration*. Clicando nele podemos notar que há dois itens criados: *Release* e *Debug*. Podemos criar nossos próprios itens.

Clique em *Compiler* e nas configurações desmarque as caixas *Optimization* e *Generation XML documentation* ou marque e desmarque outras opções a sua escolha. Em *Compiler Messages* retire alguns *Warnings* do grupo principal e em *Directories/*

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Adriano Santos que mostra como utilizar as configurações de Build no Delphi 2007.

www.devmedia.com.br/articles/viewcomp.asp?comp=5349

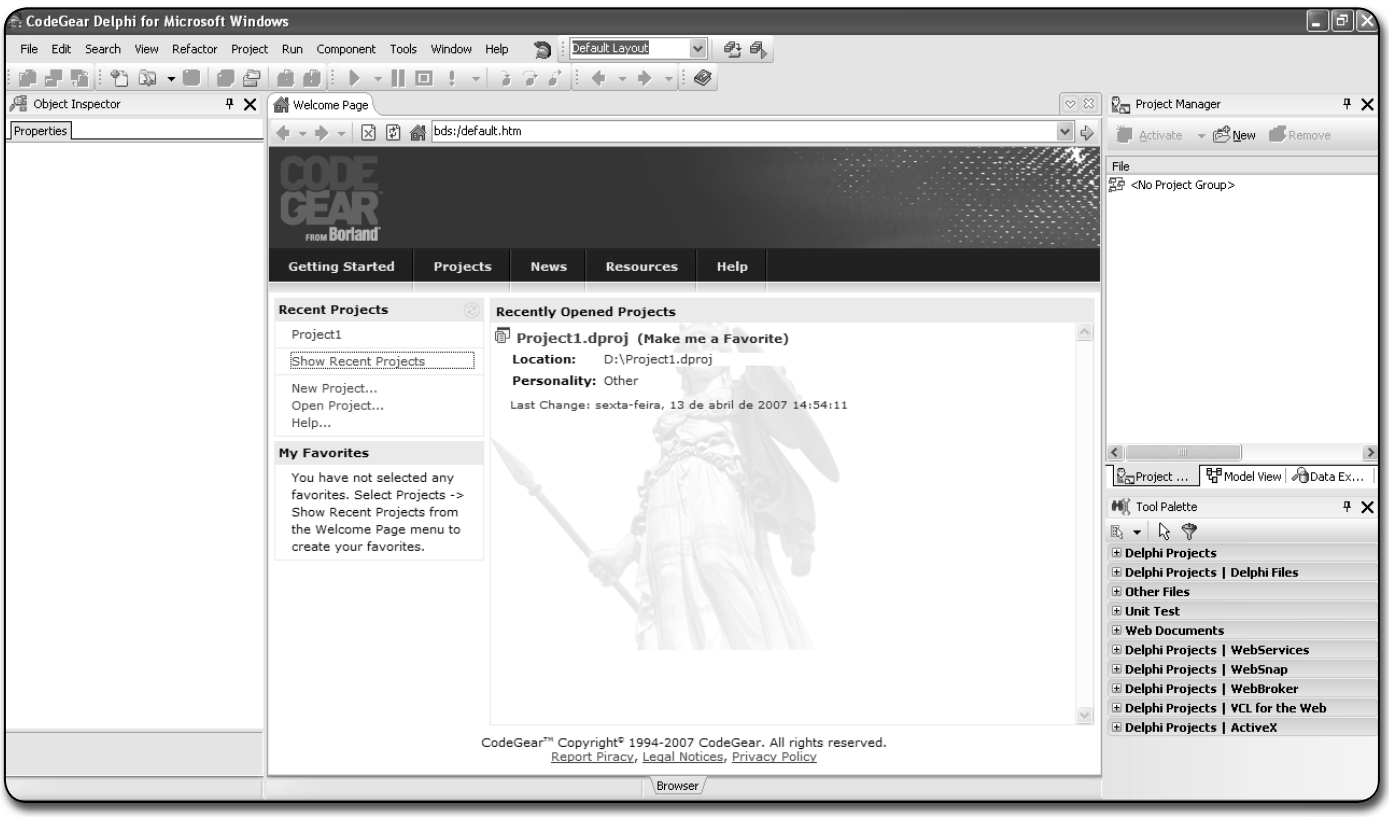


Figura 1. Home inicial do Delphi 2007

Conditionals mude os diretórios de output tais como *Output Directory* e *Unit Output Directory*.

Feito isso clique em *Save As* ao lado do *Build Configuration*. Na tela que se abre apenas digite o nome que deseja chamar essa configuração e clique OK. Repita a opção fazendo novas configurações e salve-a.

Clique agora em *Build Configuration* e alterne entre o primeiro item que criamos e qualquer outro item. Veja que as configurações são carregadas de acordo com a configuração que fizemos (**Figura 2**).

Alguns detalhes que passam despercebidos, são as novas extensões dos arquivos de projeto que passaram de BDSPROJ e BDSGROUP para DPROJ e GROUPPROJ respectivamente. Essas novas extensões são necessárias para compatibilizar os projetos com ferramentas de Build automático, tais como *MS Build* da Microsoft ou o *Ant*, aplicativo *freeware* bastante conhecido da comunidade Java.

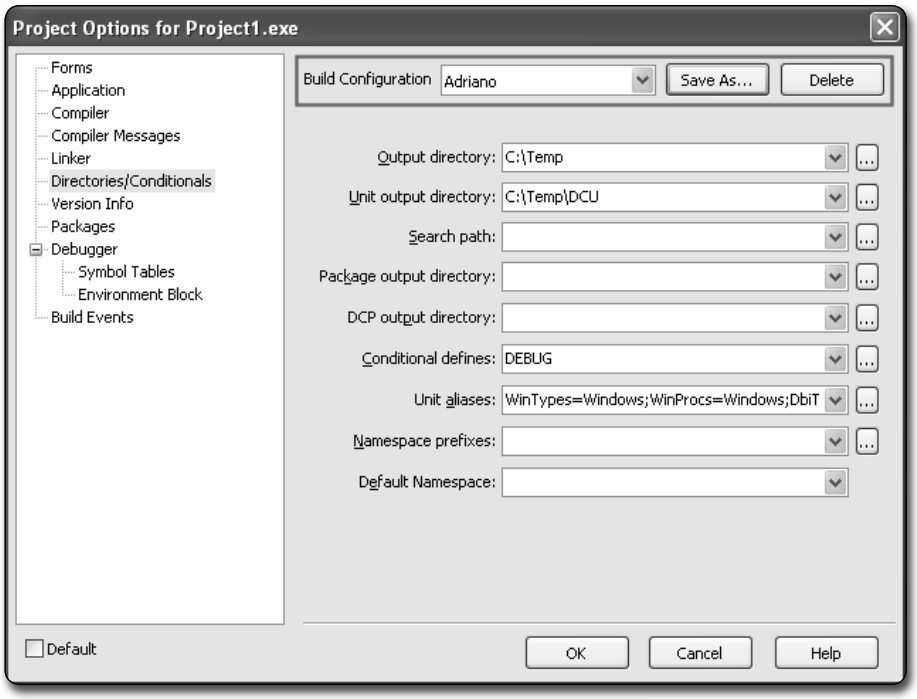


Figura 2. Build Configuration

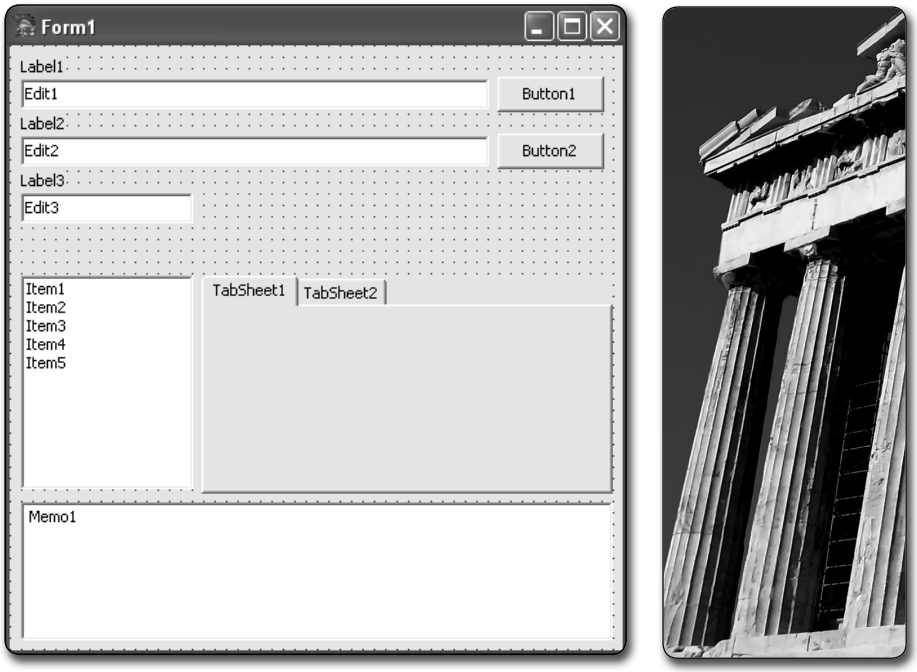


Figura 3. Exemplo de janela para teste com Vista Themes

Isso significa que ao abrir um projeto feito em Delphi 2006, por exemplo, o Delphi 2007 mostrará as mensagens *Creating a new project* e *Upgrading Project*, ambas acompanhadas dos diretórios e nomes de arquivos a serem criados e/ou atualizados.

Outro recurso de extrema importância (também nas opções do projeto) é o *Enable runtime themes* que foi adicionado para compatibilizar visualmente nossa aplicação a todos os sistemas operacionais Windows.

Entre as versões 7 e BDS 2006 tínhamos o *XP Manifest*, que fazia com que nossas aplicações ficassem com o estilo visual do Windows XP. Para isso bastava incluir o componente na tela principal e compilar o projeto.

Esse recurso agora é um opcional e é válido para qualquer Windows. Para testar o recurso *Enable runtime themes* insira alguns componentes no projeto que acabamos de criar tais como: *Buttons*, *Edits*, *Labels*, *ListBoxes* e *ComboBoxes* (**Figura 3**).

Agora entre no menu *Project>Options* e na opção *Application* marque o item *Enable runtime themes*. Compile sua aplicação e execute. Esse recurso detecta o Windows onde a aplicação está rodando e modifica o visual das telas e componentes pra se adaptar ao sistema operacional.

Dessa forma, nosso usuário não terá a sensação de estar com uma versão ultrapassada do sistema rodando em um Windows mais recente. Observe na **Figuras 4** o mesmo exemplo rodando em Windows Vista e XP.

Ainda se tratando de Windows Vista, a CodeGear introduziu e atualizou as bibliotecas de mensagens e diálogos do Delphi. A exemplo disso temos agora a paleta *Vista Dialogs* onde encontramos os componentes *FileOpenDialog*, *FileSave Dialog* e *TaskDialog*.

O *TaskDialog* certamente é a maior novidade. Com ele podemos adicionar diversos recursos aos diálogos, tais como *RadioButtons*, *ProgressBars* e botões personalizados. Crie uma nova aplicação Win32 e vamos entender um pouco melhor como isso funciona.

Adicione um *TaskDialog* da paleta *Vista Dialogs* no formulário. Clique duas vezes na propriedade *RadioButtons*. Na tela que

se abre adicione três itens e modifique a propriedade *Caption* de cada um para que fique como "Item1", "Item2" e "Item3" respectivamente.

Retornando ao componente abra a propriedade *Flags* e marque como *True* a opção *tfShowProgressBar*. Dessa forma adicionamos três *RadioButtons* e um *ProgressBar* na caixa de diálogo.

No evento *OnRadioButtonClicked* do *TaskDialog* digite o código da **Listagem 1**.

O código identifica o *RadioButton* clicado e configura a propriedade *Position* da *ProgressBar* de maneira que fique com 10, 50 e 100 por cento preenchida. Vamos a mais um exemplo agora usando a propriedade *Buttons* do componente em questão.

Clique duas vezes na propriedade e insira um novo item no editor que se abre. No item criado configure sua propriedade *Caption* para "Meu Botão Personalizado". Adicione novos itens e configure a seu gosto.

Após as devidas configurações adicione um *Button* no formulário e o faça a chamada ao método *Execute* do componente. Execute a aplicação e clique no botão para executar a caixa de diálogo. Veja na **Figura 5** a caixa de diálogo em execução no Windows Vista.

Listagem 1. Código de atualização da ProgressBar no TaskDialog

```
procedure TForm1.TaskDialog1RadioButtonClicked(Sender: TObject);
begin
  if TaskDialog1.RadioButton = TaskDialog1.RadioButtons[0] then
    TaskDialog1.ProgressBar.Position := 10;
  if TaskDialog1.RadioButton = TaskDialog1.RadioButtons[1] then
    TaskDialog1.ProgressBar.Position := 50;
  if TaskDialog1.RadioButton = TaskDialog1.RadioButtons[2] then
    TaskDialog1.ProgressBar.Position := 100;
end;
```

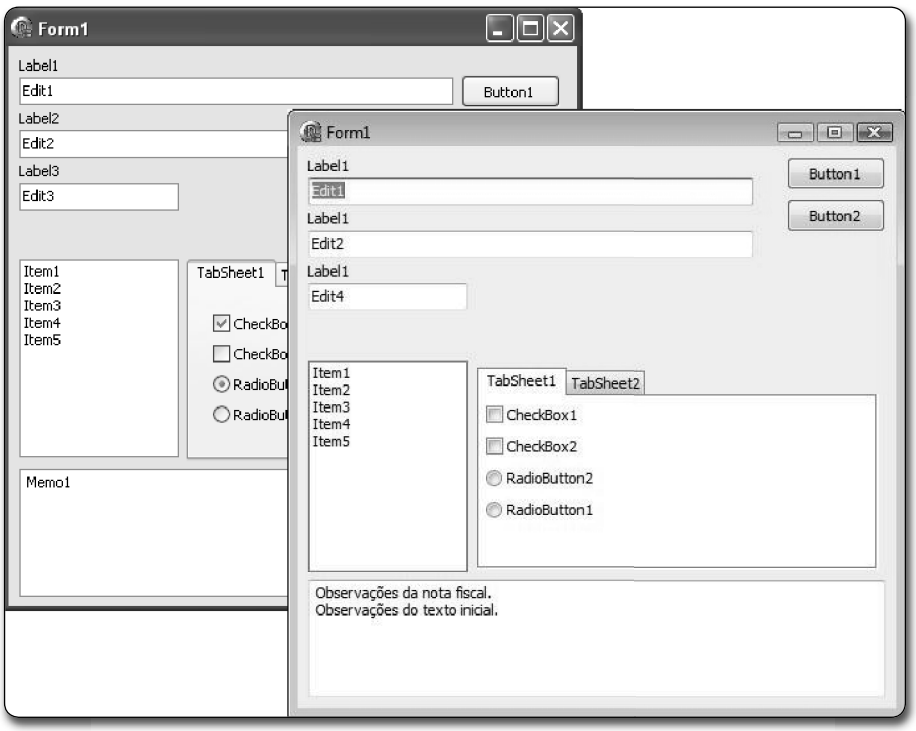


Figura 4. Utilizando Enable runtime themes no Windows Vista e XP

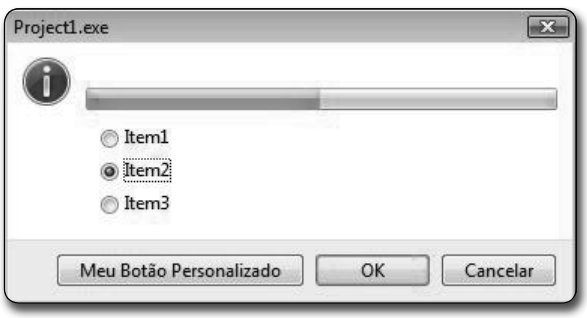


Figura 5. Exemplo de TaskDialog no Windows Vista

> **Nota:** Os componentes, obviamente, serão mostrados apenas se você estiver trabalhando com o Windows Vista.

O último teste com os componentes da paleta *Vista Dialogs* é inserirmos no formulário um *FileOpenDialog* e um *FileSaveDialog*. Insira também dois novos botões e no evento *OnClick* de cada um faça uma chama ao *Execute* de cada componente. Veja na **Figura 6** as novas caixas de diálogo sendo exibidas.

> **Nota:** Ao utilizar somente os componentes da paleta *Vista Dialogs* estamos compatibilizando nossas aplicações apenas com Windows Vista, portanto deve-se ter atenção ao escolher tais componentes.

Mais gráficos no Delphi

Até a versão BDS 2006 tínhamos apenas uma opção para geração de gráficos. Isso mudou e temos agora uma poderosa suíte de componentes que agiliza a produção de gráficos em nossas aplicações. Crie uma nova aplicação e insira os componente *SqlConnection* ("sqlConexao") e *sqlDataSet* ("sqlEmployee") da paleta *dbExpress*.

Faça a ligação do *SqlDataSet* ao *Sql Connection* usando a propriedade *Sql Connection*. Configure o *SqlConnection* para conectar-se ao banco *Employee.fdb* presente no diretório *examples* do Firebird.

Adicione também um *DataSetProvider* ("dspEmployee"), um *ClientDataSet* ("cdsEmployee") e um *DataSource* ("dtsEm-

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp
Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Adriano Santos que mostra como utilizar o Enable runtime thmes no Windows Vista, XP e 98.
www.devmedia.com.br/articles/viewcomp.asp?comp=5349

ployee") todos da paleta *Data Access*. Faça todas as ligações dos componentes para que possamos visualizar os dados. Na propriedade *CommandText* do *SqlDataSet* adicione o seguinte comando SQL:

```
SELECT * FROM JOB
```

Clique duas vezes no *SqlDataSet* e adicione todos os campos da tabela. Vamos ao gráfico. Insira um *DBChart* da paleta *TeeChart Std* no formulário e clique duas vezes nele. Na guia *Series* clique no botão *Add*, adicione uma série utilizando a opção *Bar*, criando dessa forma um gráfico do tipo barras.

Após confirmar e retornar ao formulário, adicione um *DBCrossTabSource*. Configure sua propriedade *DataSet* escolhendo o *cdsEmployee*. Na propriedade *GroupField* escolha *JOB_COUNTRY* e em *ValueField* escolha *MAX_SALARY*. Nesse ponto basta ativar o gráfico marcando a propriedade *Active* para *True*.

Repare que o gráfico já foi montado e está pronto para ser exibido (**Figura 7**).

Pra finalizar, adicione um *ButtonColor* da paleta *TeeChart Std* na tela. Com ele, vamos mudar a cor da primeira coluna do *DBChart*, pois o botão tem a característica de abrir automaticamente a caixa de diálogo de cores padrão do Windows, dessa forma podemos pegar a cor selecionada.

Vamos usá-la para substituir a cor da primeira barra. Para isso, adicione o seguinte código no evento *OnClick* do botão:

```
DBChart1.Series[0].Color :=
  ButtonColor1.SymbolColor;
```

Execute novamente o exemplo e teste o botão.

AJAX na VCL

Outra grande notícia do Delphi 2007 é a total compatibilidade do AJAX com a nova *IntraWeb*, agora chamada de *VCL for Web*. Está muito mais fácil incluir rotinas em aplicações Web sem fazer requisições desnecessárias ao servidor. Vamos criar um exemplo simples pra demonstrar como usar AJAX nessa nova versão.

Crie uma aplicação Web através do menu *File>New>Other>VCL for Web* e escolha a opção *VCL for the Web Application Wizard*. Clique em OK na próxima tela e aguarde a criação dos formulários necessários.

Na paleta de componentes adicione quatro *IWEdits*, quatorze *IWLabels* e um *IWButton*. Coloque-os no formulário principal como na **Figura 8**.

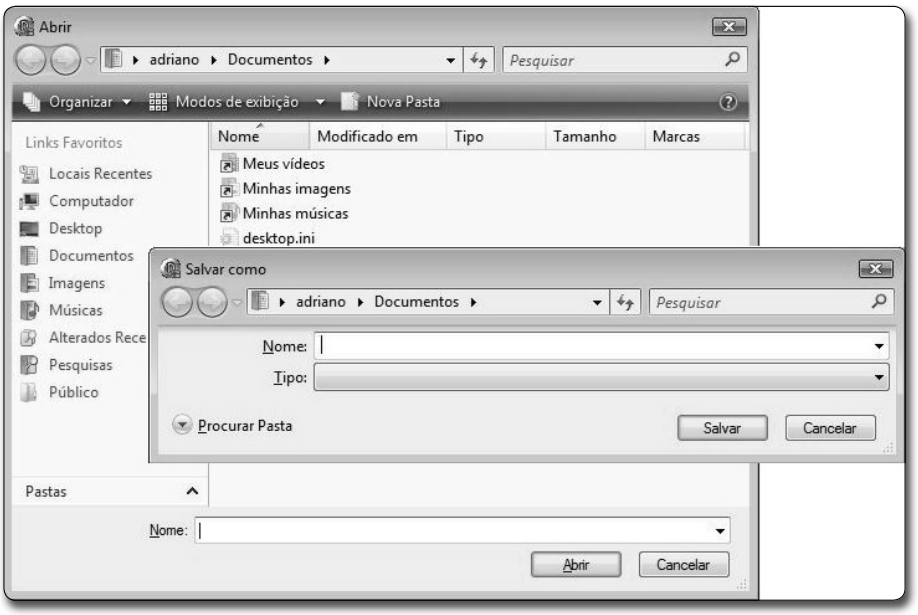


Figura 6. FileOpenDialog e FileSaveDialog no Windows Vista

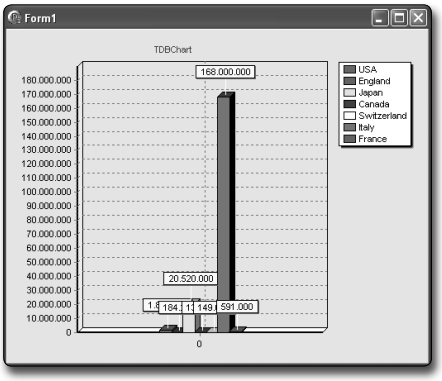


Figura 7. Gráfico usando TeeChart

Nomeie os *IWEdits* para "edtNome", "edtEndereco", "edtCidade" e "edtEstado", e mude o *Caption* de cada *IWLabel* para "Nome", "Endereço", "Cidade" e "Estado" respectivamente. No *IWButton* troque o *Name* para "btnConfirmar" e o *Caption* para "Confirmar".

Os dois *IWLabels* que sobraram, coloque-os acima do nosso formulário de cadastro com o *Caption* que preferir. Neste exemplo usamos *ClubeDelphi* no primeiro e *Exemplo de Cadastro* no segundo.

Abaixo do botão colocamos oito *Labels* sendo que os quatro à direita receberão os nomes "lblNome", "lblEndereco", "lblCidade" e "lblEstado". Esses *Labels* serão atualizados com os dados que serão digitados nos *Edits* simulando uma pré-confirmação do que foi digitado.

Nos *Labels* à esquerda apenas mude o *Caption* de cada um deles e suas configurações de fonte se necessário. Agora no evento *onAsyncClick* do *btnConfirmar* vamos fazer a codificação do exemplo fazendo com que os *Labels* do rodapé sejam atualizados. Insira o código da **Listagem 2**.

Pressione F9 e execute a aplicação. Note que ao digitar algo nos *Edits* e clicar no botão não há mais recarga da página e todos os *Labels* são atualizados, isso porque estamos utilizando o evento *OnAsyncClick* que utiliza o AJAX para atualizar a página (**Figura 9**). '

Até a versão 2006 do Delphi teríamos que criar uma série de funções *JavaScript* para nos auxiliar na criação de sites com AJAX. Sempre que for necessária a utilização de AJAX em aplicações Web, basta transferirmos o código do evento convencional para os eventos do tipo *Async*.

> **Nota:** Observe que quase todos os componentes possuem os métodos *Async* tais como *OnAsyncChange*, *OnAsyncExit* do *IWEdit*.

Compatibilidade com componentes de terceiros

E os recursos não param por ai. A CodeGear se preocupou também com uma das partes cruciais do Delphi: a compatibilidade com componentes de terceiros.

Quem usa componentes desenvolvidos por outros desenvolvedores e/ou empresas sabe que uma das primeiras preocupações quando se pensa em migrar o sistema para uma versão mais recente do Delphi é procurar ou aguardar novas verões desses componentes, ou as vezes até abandonar alguns deles.

O problema pode ser resolvido apenas recompilando o código-fonte do pacote no próprio Delphi 2007 e efetuar a instalação. Isso já faz com que o componente se compatibilize com a versão 2007 facilitando a migração de nossos sistemas.

Desde a instalação do Delphi em meu laboratório de testes, efetuei a compilação de diversas suítes tais como: *RxLib*, *LMD*, *FortesReport*, *JediVCL* e *QuickReport 4*. Todos sem a necessidade de baixar uma versão especial para Delphi 2007.

Os componentes que não possuem fonte e não estão compatibilizados com 2007 deverão passar por esse mesmo processo, mas nesse caso quem o terá que fazer é o próprio desenvolvedor e/ou empresa detentora do código-fonte.

Listagem 2. Código para atualizar os Labels

```
procedure TIWForm1.btnConfirmarClick(Sender: TObject);
begin
  lblNome.Caption := edtNome.Text;
  lblEndereco.Caption := edtEndereco.Text;
  lblCidade.Caption := edtCidade.Text;
  lblEstado.Caption := edtEstado.Text;
end;
```

Conclusão

Este artigo mostrou as principais características da nova versão do *Delphi for Win32*. Entre as novidades estão as modificações no *Project>Options* para compatibilizar as aplicações com *MSBuild* e outras ferramentas de Build automático. Criação de favoritos na *Home* para melhor gerenciamento dos projetos mais recentes.

Os novos componentes, atualização de mensagens e diálogos para compatibilizar com Windows Vista. Também foi vista a nova suíte de componentes para criação de gráficos, *TeeChart*, e a adição de eventos especiais pra lidar com *AJAX* na nova *IntraWeb*.

Para melhor entender o funcionamento e as novidades dessa versão acesse o portal da DevMedia e assista a seis vídeo-aulas mostrando as principais novidades e também como instalar a versão trial do Delphi 2007.

Um forte abraço e até a próxima. ●

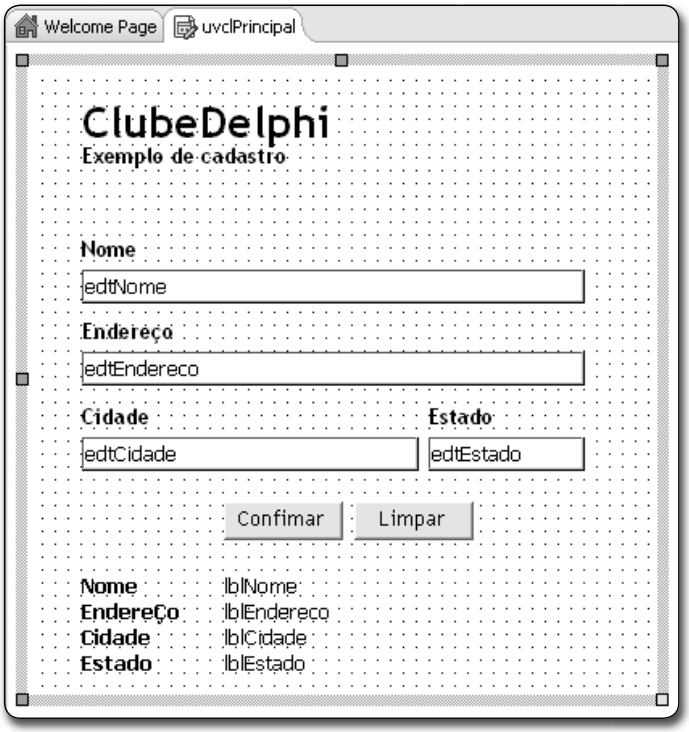


Figura 8. Exemplo de tela em VCL for Web

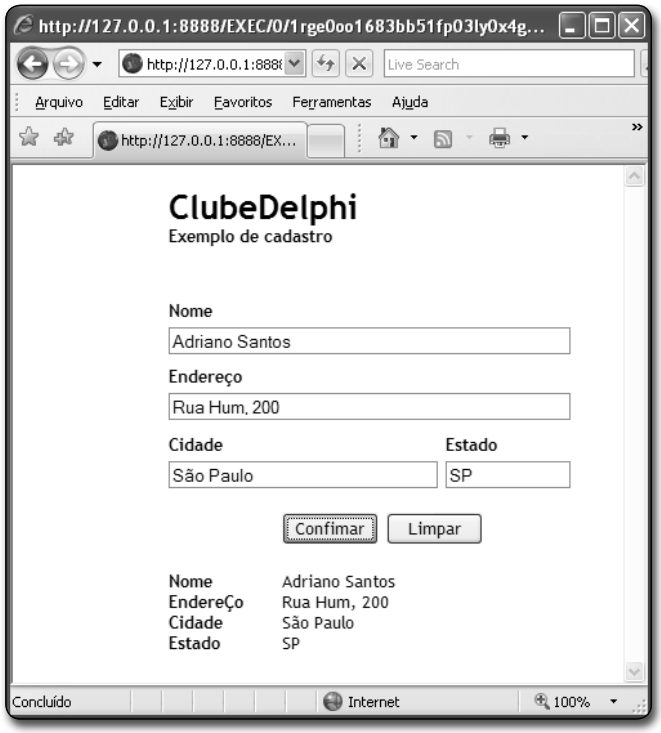


Figura 9. Aplicação Web utilizando AJAX

+ de 80.000 membros cadastrados
+ de 15.000 exemplos com fontes
+ de 900 apostilas
+ de 4.000 dicas
Fórum Delphi
Artigos
TOTALMENTE GRÁTIS
www.delphi.eti.br
Um dos maiores sites de apoio a desenvolvedores Delphi do Brasil!!!

SOAP - Aprenda técnicas avançadas em um exemplo passo a passo – Parte 2

Michael Benford
(michael@devmedia.com.br)
é Acadêmico do curso de matemática da Universidade Federal Fluminense e desenvolvedor Delphi e .NET da QSi, Qualidade em Sistemas de Informação. Programa em Delphi há 5 anos, desenvolvendo aplicações cliente/servidor, multicamadas, Web, utilitários de uso geral, componentes, experts e aplicativos não-comerciais. É colunista do portal da DevMedia, articulista e membro da comissão editorial da revista Clube-Delphi. Palestrante da primeira edição do Developerís Webdays e da BorCon Brasil 2006.

Na primeira parte deste artigo, começamos a construir nosso disco virtual, um aplicativo onde o usuário poderá enviar arquivos para um servidor remoto, listá-los e recuperá-los posteriormente. Criamos a interface gráfica do cliente, vimos um pouco sobre autenticação com SOAP, exceções remotas, envio e recebimento de SOAP Headers e como rastrear as chamadas ao servidor. Nesta edição, veremos gerenciamento de sessão, envio de arquivos via SOAP, compactação e muito mais.

Gerenciamento de sessão

O mecanismo de autenticação que implementamos funciona, mas não está completo. Ele retorna sempre o mesmo identificador de sessão, definido de forma *hard-coded* no código do servidor, o que não é, obviamente, uma solução segura. Precisamos gerar um valor que seja único entre todos os usuários que podem vir a se autenticar.

Aqui temos um problema: Web Service são por natureza *stateless*, o que significa que nenhuma informação é persistida entre as requisições processadas. Mas o servidor precisa "lembrar" dos identificadores gerados, para poder comparar com os fornecidos pelos clientes. Como conseguir isso? Uma alternativa é controlar sessões.

A implementação do protocolo SOAP disponível com o Delphi não fornece nenhum mecanismo de persistência nativo. Dessa forma, precisamos criar um. Atente então para um detalhe importante: nosso Web Service reside em uma DLL ISAPI, que é carregada pelo IIS na primeira requisição recebida pelo servidor.

Essa DLL fica no contexto do processo *DLLHost.exe*, também conhecido como *COM Surrogate*. E ela ficará lá, em memória, até ser descarregada, como você fez algumas vezes ao longo desse artigo. Mas Web Service não são *statesless*?

Sim, são! Abra novamente *uDiscoVirtualImpl* e observe a classe *TDiscoVirtual*. Esse é o nosso Web Service. É essa classe

que é criada e destruída automaticamente a cada execução remota. O resto da unit permanece intacto, carregado na memória. Confuso? Vamos implementar para você entender melhor.

Selecione o projeto do servidor e adicione a ele um Data Module. Chame-o de "TdmSession" e salve o arquivo com o nome "uSessionDataModule". Clique agora no menu *Project* e escolha *View Source*. Remova a seguinte linha de código:

```
Application.CreateForm(tdmSession,dmSession);
```

Fizemos isso porque criaremos manualmente o Data Module. Adicione um *ClientDataSet*. Dê um duplo clique sobre ele para abrir o editor de campos e insira os seguintes *Fields*:

1. SessionID: Tipo *String*, tamanho 38;
2. LastAccess: Tipo DateTime;
3. UserName: Tipo *String*, tamanho 10;
4. UserID: Tipo Integer.

Clique de direita sobre o *ClientDataSet* e escolha *Create DataSet*. Pronto, já temos um *DataSet* para armazenar os identificadores de sessão gerados para cada usuário. Vamos então criar alguns métodos para gerenciar sessões. Adicione as linhas da **Listagem 1** na seção *public* da classe *TdmSessions*:

Implemente agora todos esses métodos (*Ctrl+Shift+C*) com o código da **Listagem 2**.

Esses métodos servem para:

- Criar um novo registro no *ClientDataset* de sessões, retornando um identificador único (*NewSession*);
- Remover uma sessão do *DataSet* (*RemoveSession*);
- Verificar se existe uma sessão com o identificador informado ou se o usuário está há muito tempo sem realizar uma requisição (*IsValidSession*);
- Retornar o ID do usuário de uma determinada sessão (*GetUserID*) e para atualizar a data/hora do último acesso do usuário (*UpdateLastAccess*).

Observe o uso de *critical sections* para sincronizar o acesso ao código, pois o Web Service pode receber muitas solicitações simultâneas. Essas seções críticas vão garantir que apenas uma chamada por vez execute o código entre os métodos *Enter* e *Leave*, preservando a aplicação em um estado consistente.

Listagem 1. Métodos de TdmSessions

```
function NewSession(const UserID: Integer;
  UserName: string): string;
procedure RemoveSession(const SessionID: string);
function IsValidSession(const SessionID: string): Boolean;
function GetUserID(const SessionID: string): Integer;
procedure UpdateLastAccess(const SessionID: string);
```

Listagem 2. Métodos de manipulação de sessão

```
function TdmSessions.NewSession(const UserID: Integer; UserName: string): string;
var
  SessionID: TGUID;
begin
  CriticalSection.Enter;
  try
    if not cdsSessions.Active then cdsSessions.Open;
    CreateGUID(SessionID);
    Result := GuidToString(SessionID);
    cdsSessions.Insert;
    cdsSessionsSessionID.AsString := Result;
    cdsSessionsStartDateTime.AsDateTime := Now;
    cdsSessionsLastAccess.AsDateTime := Now;
    cdsSessionsUserName.AsString := UserName;
    cdsSessionsUserID.AsInteger := UserID;
    cdsSessions.Post;
  finally
    CriticalSection.Leave;
  end;
end;

function TdmSessions.IsValidSession(const SessionID: string): Boolean;
begin
  CriticalSection.Enter;
  try
    Result := cdsSessions.Locate('SessionID', SessionID, []);
    if Result then
      Result := MinutesBetween(cdsSessionsLastAccess.AsDateTime, Now) < 1;
  finally
    CriticalSection.Leave;
  end;
end;
function TdmSessions.GetUserID(const SessionID: string): Integer;
begin
  CriticalSection.Enter;
  try
    Result := 0;
    if IsValidSession(SessionID) then
      Result := cdsSessionsUserID.AsInteger;
  finally
    CriticalSection.Leave;
  end;
end;
procedure TdmSessions.RemoveSession(const SessionID: string);
begin
  CriticalSection.Enter;
  try
    if IsValidSession(SessionID) then
      cdsSessions.Delete;
  finally
    CriticalSection.Leave;
  end;
end;
procedure TdmSessions.UpdateLastAccess(const SessionID: string);
begin
  CriticalSection.Enter;
  try
    if IsValidSession(SessionID) then
    begin
      cdsSessions.Edit;
      cdsSessionsLastAccess.AsDateTime := Now;
      cdsSessions.Post;
    end;
  finally
    CriticalSection.Leave;
  end;
end;
```

Para usá-las, porém, precisamos realizar mais algumas coisas. Inclua a unit *SyncObjs* à cláusula *uses* e declare a variável *CriticalSection* na seção *var* da mesma, como mostrado a seguir:

```
var
  dmSessions: TdmSessions;
  { Adicione essa linha }
  CriticalSection: TCriticalSection;
implementation
```

Por fim, implemente as seções *initialization* e *finalization* da unit com o código a seguir:

```
initialization
  CriticalSection := TCriticalSection.Create;
finalization
  CriticalSection.Free;
```

Volte ao arquivo *uDiscoVirtualImpl* e adicione a linha a seguir à mesma seção

var mencionada anteriormente:

```
var
  Sessions: TdmSessions;
implementation
```

E, por último, novamente nas seções *initialization* e *finalization* da unit, inclua o seguinte código (apenas as linhas destacadas):

```
initialization
  { Adicione essa linha }
  Sessions := TdmSessions.Create(nil);
finalization
  Sessions.Free; { E essa também}
```

Muito bem, já podemos fazer uso do mecanismo de sessão. Então, retorne aos métodos *Login* e *CheckHeader* e modifique-os com o código da **Listagem 3**. Também nessa listagem está o código do *Logout*, que deve ser implementado.

Descarregue a DLL do servidor no IIS e recompile o projeto. Alterne para o projeto do cliente e abra o evento *OnClick* do *Login*. Crie um desvio com *else* para o primeiro *if* existente (*if FLoggedOut then*) e adicione o código da **Listagem 4**.

Observe que estamos tratando as exceções vindas do *Logout*, para evitar que a aplicação não atualize os controles na tela e o usuário não possa realizar o login novamente. Execute agora o cliente, efetue o login e observe na barra de status o novo código de identificação recebido (**Figura 1**).

Teste as demais funcionalidade implementadas, como informar um nome de usuário inexistente ou uma senha inválida. Se você demorar mais do que um minuto para realizar o *logout*, receberá uma exceção vinda do *IsSessionValid*, que determina a sessão como expirada se esse tempo transcorrer.

O valor de um minuto foi definido propositalmente para ser possível testar o recurso. Em produção você deve escolher um valor mais perto da realidade da aplicação.

Listagem 3. Criando e verificando sessões

```
procedure TDiscoVirtual.Login(const UserName, Password: string);
var
  UserID: Integer;
  SessionID: string;
  Header: TAuthenticationHeader;
begin
  with TDataAccess.Create(nil) do
  try
    cdsUsers.Params.ParamByName('NOME').AsString := UserName;
    cdsUsers.Open;
    if cdsUsers.RecordCount = 0 then
      raise EUserNotFound.CreateFmt('Usuário %s não foi encontrado', [UserName]);
    if not SameStr(cdsUsersSENHA.AsString, Password) then
      raise EInvalidPassword.Create('A senha informada é inválida');
    UserID := cdsUsersID.AsInteger;
  finally
    Free;
  end;
  SessionID := Sessions.NewSession(UserID, UserName);
  Header := TAuthenticationHeader.Create;
  Header.SessionID := SessionID;
  (Self as ISOAPHeaders).Send(Header);
end;
procedure TDiscoVirtual.CheckHeader(var SessionID: string);
var
  Header: TAuthenticationHeader;
begin
  (Self as ISOAPHeaders).Get(TAuthenticationHeader, TSOAPHeader(Header));
  if not Assigned(Header) then
    raise ERemotableException.Create('Cabeçalho de identificação não encontrado');
  if Sessions.IsValidSession(Header.SessionID) then
  begin
    SessionID := Header.SessionID;
    Sessions.UpdateLastAccess(SessionID);
  end
  else
    raise ERemotableException.Create('Identificador de sessão inválido. '+
      'Por favor efetue o login novamente');
end;
procedure TDiscoVirtual.Logout;
var
  SessionID: string;
begin
  CheckHeader(SessionID);
  Sessions.RemoveSession(SessionID);
end;
```

Listagem 4. Alterando quando o usuário não for autenticado

```
(...)
  else
  begin
    try
      PrepareHeader;
      AppServer.Logout;
    except
      on E: ERemotableException do
        MessageBox(E.Message, MB_OK + MB_ICONERROR);
    end;
    FLoggedOut := True;
    btnLogin.Caption := 'Login';
    StatusBar.Panels[0].Text := 'Não autenticado';
    StatusBar.Panels[1].Text := '';
    lstRemoteFiles.Items.Clear;
    edtServerURL.Enabled := True;
  end;
end;
```

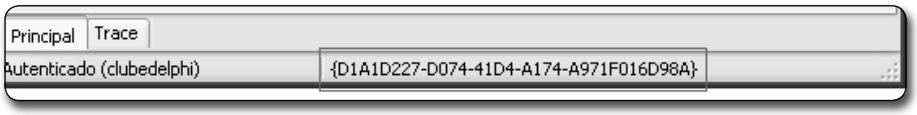


Figura 1. Identificador único de sessão recebido do servidor

Enviando arquivos via SOAP

Prosseguindo com o desenvolvimento da nossa aplicação, vamos ver agora como enviar arquivos através do protocolo SOAP. Então, sem demoras, selecione o projeto do servidor e abra o Data Module criado anteriormente.

Adicione os seguintes componentes: dois *SQLDataSets*, dois *DataSetProviders* e dois *ClientDataSets*. Configure o relacionamento entre cada trio e nomeio-os conforme mostrado na **Figura 2**.

Configure a propriedade *CommandText* do *dsFileInsert* para "select ID, ID_USUARIO, NOME_ARQUIVO, TAMANHO, DATA_HORA , ARQUIVO from ARQUIVO where ID_USUARIO = :ID_USUARIO".

Dê um duplo clique sobre o componente para abrir o editor de campos, clique de direita sobre ele e selecione *Add all fields*. Em seguida, configure a propriedade *Required* do campo ID para *False*. Repita o mesmo procedimento para o *cdsFileInsert*.

Para o *dsFileSelect*, altere a propriedade *CommandText* para "select ID_USUARIO, NOME_ARQUIVO, TAMANHO, DATA_HORA from ARQUIVO where ID_USU-

ARIO = :ID_USUARIO". Abra o editor de campos do *cdsFileSelect* e adicione todos os *Fields* retornados na consulta.

Você deve estar pensando porque estamos usando dois *DataSets* ao invés de apenas um, já que as *queries* são praticamente iguais. Pois o detalhe é exatamente esse: elas são **apenas** parecidas. Observe que o segundo comando SELECT não retorna o campo ARQUIVO, onde os arquivos enviados serão armazenados.

Isso é proposital: se carregássemos para a memória todos os arquivos que o usuário possa ter enviado para o servidor, estaríamos literalmente destruindo o desempenho da aplicação. Mais: também faríamos que todos trafegassem pela rede em uma simples listagem, por exemplo, o que obviamente é desnecessário.

Explicado esse ponto, abra *uDiscoVirtualImpl* e localize o *UploadFile*. Preste atenção no seu parâmetro *FileData*: ele é do tipo *TByteDynArray*, que é um *array* dinâmico de *bytes*. Podemos armazenar nele qualquer coisa serializável para *bytes*, e isso inclui, é claro, arquivos.

Implemente então o código do *UploadFile* como mostrado na **Listagem 4**.

Note que estamos fazendo uso pela primeira vez da segunda versão do *CheckHeader*, para obter o identificador de sessão recebido no cabeçalho da mensagem SOAP. Precisamos dele para chamar o *GetUserID*, do Data Module de sessões, a fim de conseguir o identificador do usuário no banco de dados, necessário para inserir um registro na tabela de arquivos.

Por fim, repare que estamos convertendo o parâmetro *FileData*, que é do tipo *TByteDynArray*, em um *TStream*, através do *ByteArrayToStream*, da classe *ConvertEngine*. Essa classe, escrita por mim, encontra-se na unit *Tools*, disponibilizada juntamente com os fontes da aplicação.

Mesmo seu código tendo apenas duas linhas, ele não será publicado aqui novamente por questões de espaço, mas é recomendável que você o estude para aprender como a conversão funciona.

Nota: Estou considerando que o leitor saiba lidar com os componentes de acesso a dados do Delphi, bem como a tecnologia *dbExpress*. Dessa forma, não comentarei as demais linhas de código da **Listagem 4**.

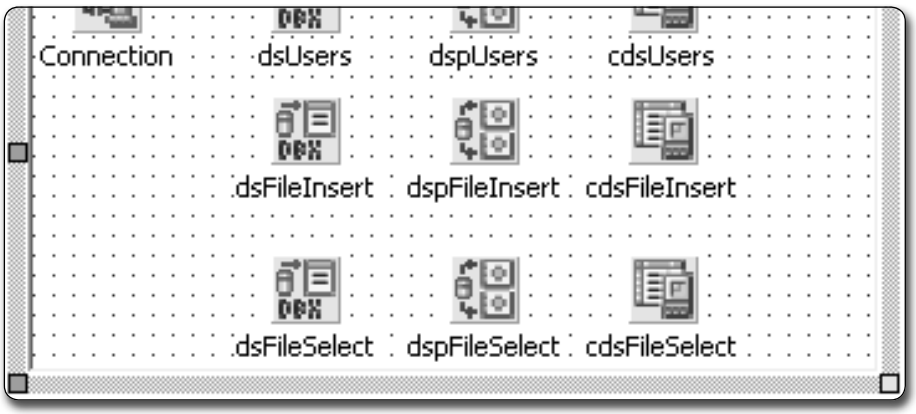


Figura 2. Novos componentes de acesso a dados

Listagem 5. Enviando arquivos com TByteDynArray e SOAP

```
procedure TDiscoVirtual.UploadFile(const FileName: string; FileData: TByteDynArray);
var
  FileStream: TMemoryStream;
  SessionID: string;
begin
  CheckHeader(SessionID);
  FileStream := TMemoryStream.Create;
  try
    ConvertEngine.ByteArrayToStream(FileData, FileStream);
    with TDataAccess.Create(nil) do
    try
      cdsFileInsert.Open;
      cdsFileInsert.Insert;
      cdsFileInsertID_USUARIO.AsInteger := Sessions.GetUserID(SessionID);
      cdsFileInsertNOME_ARQUIVO.AsString := FileName;
      cdsFileInsertTAMANHO.AsInteger := FileStream.Size;
      cdsFileInsertDATA_HORA.AsDateTime := Now;
      cdsFileInsertARQUIVO.LoadFromStream(FileStream);
      cdsFileInsert.Post;
      cdsFileInsert.ApplyUpdates(0);
    finally
      Free;
    end;
  finally
    FileStream.Free;
  end;
end;
```

Com o *UploadFile* pronto, vamos implementar agora o seu oposto: *DownloadFile*. Portanto, localize-o dentro da unit e digite o código da **Listagem 5**.

Aqui fazemos o inverso do método anterior: localizamos o arquivo pelo nome, salvamos o campo BLOB para um *stream* e o convertemos para o tipo *TByteDynArray*, novamente através da classe *ConvertEngine*, com o *StreamToByteArray*.

Agora só precisamos invocar esses métodos do cliente. Recompile então o servidor e alterne para o projeto *DiscoVirtualClient*. Codifique o evento *OnClick* do *Enviar* como mostrado na **Listagem 6**.

Não há mistérios aqui. Simplesmente carregamos o arquivo para um *stream* (através da classe *TFileStream*), e depois fizemos uso novamente da classe *ConvertEngine*. O resto é exatamente igual ao que vimos na implementação do servidor.

Observe novamente que chamamos o *PrepareHeader* antes de execução remota. Como mencionado anteriormente, precisamos fazer isso antes de cada chamada ao servidor para enviar junto como envelope do SOAP o cabeçalho contendo o identificador de sessão recebido pelo método Login.

Antes de testarmos o envio de arquivos, precisamos implementar o evento *OnClick* do *Procurar*. Então, dê um duplo clique sobre ele e digite o código da **Listagem 7**.

Fizemos uso da função nativa do Delphi, *PromptForFileName*. Ela é muito útil quando precisamos exibir ao usuário uma caixa de diálogo Abrir ou Salvar, evitando o *overhead* da criação dos componentes *OpenDialog* e *SaveDialog*.

Entretanto, a função tem suas limitações: ela não permite que você configure vários aspectos da caixa de diálogo, disponíveis através da propriedade *Options* dos componentes mencionados. Nesse caso não há outra escolha a não ser usá-los.

Dica: Confira no help do Delphi o significado de todos os parâmetros de *PromptForFileName*.

Execute a aplicação e realize o login no servidor. Em seguida, selecione um arquivo no seu computador e clique no *Enviar*. Dependendo do tamanho, o processo demorará muito ou pouco. Agora que sabemos como enviar arquivos através do protocolo SOAP, precisamos exibí-los ao usuário, para que ele possa selecionar um e realizar o seu download.

Veremos isso no próximo tópico.

Nota: Há ainda uma outra forma de trafegar arquivos via SOAP. Trata-se da classe *TSOAPAttachment*. Porém, os arquivos enviados dessa forma são codificados apenas no padrão MIME, o que pode limitar as aplicações do Web Service. Se quiser saber mais sobre essa classe, consulte a seção *Links*.

Trafegando DataSets no protocolo SOAP

Acabamos de ver como trafegar *arrays* de *bytes* entre o cliente e o servidor, e como dito, podemos enviar qualquer tipo de dado através desse método, basta serializá-lo. Em alguns casos, porém, pode não ser tão fácil realizar essa conversão. Para enviar um *DataSet*, por exemplo, há uma forma muita mais simples e pronta.

Se o leitor já tiver alguma experiência com SOAP, deve conhecer a interface *IAppSOAPServer*. É ela que torna possível conectar um *ClientDataSet* a um provider remoto, exibir e atualizar seus dados. Isso é feito através do método remoto *GetRecords*, que retorna um *OleVariant*.

E se você já observou, a propriedade *Data* da classe *TClientDataset* é do mesmo tipo (acredite, não é mera coincidência). Dessa forma, é perfeitamente possível trafegar todo o conteúdo de um *DataSet* através desse tipo de dado.

Voltando para a prática, selecione o projeto do servidor, abra *uDiscoVirtualImpl* e localize o *GetFileList*. Codifique-o como mostrado na **Listagem 8**.

Esse método é auto-explicativo, exceto por um detalhe: estamos fazendo uso do *cdsFileSelect*, que não possui o campo BLOB do arquivo. Isso é proposital, pois não queremos carregá-lo para a memória, sobrecarregando desnecessariamente o servidor.

Descarregue a DLL no IIS e recompile o projeto. Selecione o projeto *DiscoVirtualClient* e insira um *ClientDataSet* no formulário. Chame-o de "cdsFiles". Vá então para a seção *private* da classe do formulário e crie um método chamado "DoGetFileList".

Pressione *Ctrl+Shift+C* e digite o código da **Listagem 9**.

Vamos entender o que fizemos. A primeira linha chama o *BeginUpdate*, da propriedade *Items* da classe *TListView*. Juntamente com *EndUpdate*, ele deve ser executado sempre antes que o controle sofra grandes inclusões e/ou exclusões, para evitar efeitos visuais, como *flickering*.

Depois, como de costume, preparamos o cabeçalho da mensagem com *PrepareHeader* e invocamos o método remoto *GetFileList*, atribuindo o seu retorno diretamente à propriedade *Data* do *ClientDataset*, preenchendo-o, dessa forma.

Em seguida, iteramos pelos registros recebidos e os adicionamos ao *ListView*. Há duas funções no código que merecem ser comentadas: *GetFileType* e *AddFileIcon*. A primeira recupera a descrição do tipo do arquivo, baseado em sua extensão.

Já a segunda extrai o ícone do programa vinculado à extensão informada e o inclui no *ImageList*, a fim de ser utilizado no *ListView*

Listagem 5. Recebendo arquivos com TByteDynArray e SOAP

```
function TDiscoVirtual.DownloadFile(const FileName: string): TByteDynArray;
var
  FileStream: TMemoryStream;
  SessionID: string;
begin
  CheckHeader(SessionID);
  FileStream := TMemoryStream.Create;
  try
    with TDataAccess.Create(nil) do
    try
      cdsFileInsert.Params.ParamByName('ID_USUARIO').AsInteger := Sessions.GetUserID(SessionID);
      cdsFileInsert.Open;
      if not cdsFileInsert.IsEmpty and
        cdsFileInsert.Locate('NOME_ARQUIVO', FileName, [loCaseInsensitive]) then
      begin
        cdsFileInsertARQUIVO.SaveToStream(FileStream);
      end;
    finally
      Free;
    end;
    ConvertEngine.StreamToByteArray(FileStream, Result);
  finally
    FileStream.Free;
  end;
end;
```

Listagem 6. Evento OnClick do botão Enviar

```
procedure TfrmMain.btnUploadFileClick(Sender: TObject);
var
  FileStream: TFileStream;
  FileData: TByteDynArray;
begin
  Screen.Cursor := crHourGlass;
  try
    FileStream := TFileStream.Create(edtFileName.Text, fmOpenRead or fmShareDenyWrite);
    try
      ConvertEngine.StreamToByteArray(FileStream, FileData);
      PrepareHeader;
      AppServer.UploadFile(ExtractFileName(edtFileName.Text), FileData);
    finally
      FileStream.Free;
    end;
  finally
    Screen.Cursor := crDefault;
  end;
end;
```

Listagem 7. Evento OnClick do botão Procurar

```
procedure TfrmMain.btnBrowseForFileClick(Sender: TObject);
var
  FileName: string;
begin
  if PromptForFileName(FileName,
    'Todos os arquivos|*.*', '',
    'Procurar arquivo') then
    edtFileName.Text := FileName;
end;
```

(não se esqueça de apontar a propriedade *SmallImages* do *ListView* para o *ImageList*).

O código de ambas não está publicado, mas encontra-se disponível juntamente com os fontes da aplicação.

> **Nota:** Poderíamos simplesmente exibir os dados em um *DBGrid*, o que seria de fato muito mais simples. Porém, para dar um acabamento mais profissional à solução, optei por utilizar um *ListView*.

Muito bem. Agora volte ao evento *OnClick* do *Login* e substitua a linha *AppServer.GetFileList* por *DoGetFileList*. Já no mesmo evento do *Enviar*, adicione uma chamada à função recém-criada logo após a invocação do método remoto *UploadFile*.

Execute a aplicação, envie alguns arquivos para o servidor e veja a listagem remota sendo exibida dentro do *ListView* (**Figura 3**).

Resta-nos agora apenas implementar o *Salvar como*. Então, dê um duplo clique sobre ele e digite o código da **Listagem 10**.

Mais uma vez, não há nada novo aqui. Pegamos o nome do arquivo selecionado no *ListView*, abrimos uma caixa de diálogo *Salvar* (usando *PromptForFileName*), executamos o método remoto *DownloadFile* e finalmente convertemos o *array* de *bytes* em um *stream*, através da classe *ConvertEngine*.

Recompile a aplicação para testar a nova funcionalidade.

Excluindo os arquivos remotos

O último recurso a ser implementado em nossa aplicação é a exclusão dos arquivos no servidor de aplicação. Então, selecione o projeto *DiscoVirtualServer*, abra *uDiscoVirtualImpl* e localize o *DeleteFile*.

Implemente-o como mostrado na **Listagem 11**.

Note que estamos usando o *cdsFileSelect*, pela mesma razão explicada no tópico *Trafegando DataSets no protocolo SOAP*. Recompile o projeto do servidor e alterne em seguida para o cliente. Implemente então o evento *OnClick* do *Excluir* como mostrado na **Listagem 12**.

Exibindo o progresso de envio e recebimento

O *HTTPRIO* e, por herança, o nosso *HTTPRIOex*, possui dois eventos para capturar o progresso do envio e recebi-

Listagem 8. Retornando a lista de arquivos remotos

```
function TDiscoVirtual.GetFileList: OleVariant;
var
  SessionID: string;
begin
  CheckHeader(SessionID);
  with TDataAccess.Create(nil) do
  try
   cdsFileSelect.Params.ParamByName('ID_USUARIO').AsInteger := Sessions.GetUserID(SessionID);
    cdsFileSelect.Open;
    Result := cdsFileSelect.Data;
  finally
    Free;
  end;
end;
```

Listagem 9. Preenchendo o ListView de arquivos remotos

```
procedure TfrmMain.DoGetFileList;
var
  FileName, FileExtension, FileType, FileDateTime: string;
  FileSize, Icon: Integer;
begin
  lstRemoteFiles.Items.BeginUpdate;
  try
    lstRemoteFiles.Items.Clear;
    PrepareHeader;
    cdsFiles.Data := AppServer.GetFileList;
    cdsFiles.First;
    while not cdsFiles.Eof do
    begin
      FileName := cdsFiles.FieldByName('NOME_ARQUIVO').AsString;
      FileExtension := ExtractFileExt(FileName);
      FileSize := cdsFiles.FieldByName('TAMANHO').AsInteger;

      { Convertemos o tamanho para kilobytes }
      if FileSize < 1024 then FileSize := 1024;

      FileType := GetFileType(FileExtension);
      FileDateTime := FormatDateTime('dd/mm/yyyy hh:nn:ss', cdsFiles.FieldByName(
        'DATA_HORA').AsDateTime);

      Icon := AddFileIcon(FileExtension);
      with lstRemoteFiles.Items.Add  do
      begin
        Caption := FileName;
        SubItems.Add(Format('%d KB', [FileSize div 1024]));
        SubItems.Add(FileType);
        SubItems.Add(FileDateTime);
        SubItems.Add(FileExtension);
        ImageIndex := Icon;
      end;
      cdsFiles.Next;
    end;
    cdsFiles.Close;
  finally
    lstRemoteFiles.Items.EndUpdate;
  end;
end;
```

Listagem 10. Baixando os arquivos do servidor

```
procedure TfrmMain.btnDownloadFileClick(Sender: TObject);
var
  FileStream: TFileStream;
  FileData: TByteDynArray;
  RemoteFileName, LocalFileName: string;
begin
  if not Assigned(lstRemoteFiles.Selected) then
    Exit;
  RemoteFileName := lstRemoteFiles.Selected.Caption;
  LocalFileName := RemoteFileName;
  if PromptForFileName(LocalFileName, 'Todos os arquivos|*.*', '', 'Salvar como', '', True)
then
  begin
    Screen.Cursor := crHourGlass;
    try
      PrepareHeader;
      FileData := AppServer.DownloadFile(RemoteFileName);
      FileStream := TFileStream.Create(LocalFileName, fmCreate or fmShareExclusive);
      try
        ConvertEngine.ByteArrayToStream(FileData, FileStream);
      finally
        FileStream.Free;
      end;
    finally
      Screen.Cursor := crDefault;
    end;
  end;
end;
```

mento de pacotes SOAP: *OnPostingData* e *OnReceivingData*. Na verdade, esse eventos pertencem à classe *THTTPReqResp*, que é composta por *THTTPRIO*.

Vamos então ver como usá-los. Adicione um novo formulário ao projeto, chame-o de "frmProgress" e salve sua unit com o nome "uFrmProgress". Em seguida, configure as propriedade *BorderStyle* e *Position* para *bsDialog* e *poScreenCenter*, respectivamente. Insira então um *Label* e um *Progressbar* (*Win32*).

Dê ao primeiro o nome de "StatusLabel", e ao segundo, "Progressbar". Por fim, disponha-os visualmente como mostrado na **Figura 4**.

Implemente agora o evento *OnCreate* do formulário com o seguinte código:

```
Height := ClientHeight;
SetWindowLong(Handle, GWL_STYLE,
GetWindowLong(Handle,
  GWL_STYLE) and not WS_CAPTION);
```

Isso removerá a barra de títulos da janela. Salve tudo e vá para o formulário principal da aplicação. Adicione então dois métodos à seção *private* da classe *TfrmMain*, chame-os de "ShowProgressDialog" e "HideProgressDialog", e implemente-os (Ctrl+Shift+C) como mostrado na **Listagem 13**.

Muito bem. Implemente agora os eventos *OnPostingData* e *OnReceivingData*, do *HTTPRIOEx*, como mostrado na **Listagem 14**.

Volte ao *OnClick* do *Enviar* e adicione as linhas destacadas na **Listagem 15**.

Faça o mesmo com o *Salvar como*. Compile a aplicação, execute-a, envie arquivos e receba arquivos para/de o servidor. Talvez você precise escolher arquivos grandes para ver com clareza a caixa de progresso sendo exibida (**Figura 5**).

Se a barra de progresso que apareceu quando você realizou o download de um arquivo não se comportou como o esperado, não foi erro seu. Há um *bug* na implementação do *HTTPReqResp*, que não atualiza corretamente os valores repassados ao evento *OnReceivingData*.

Para corrigi-lo, abra a unit *SOAPHTTPTrans* (digite o nome da unit no *Code Editor*

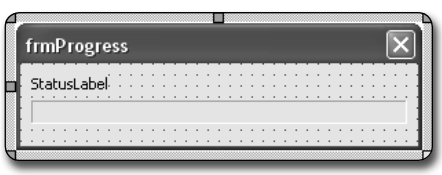
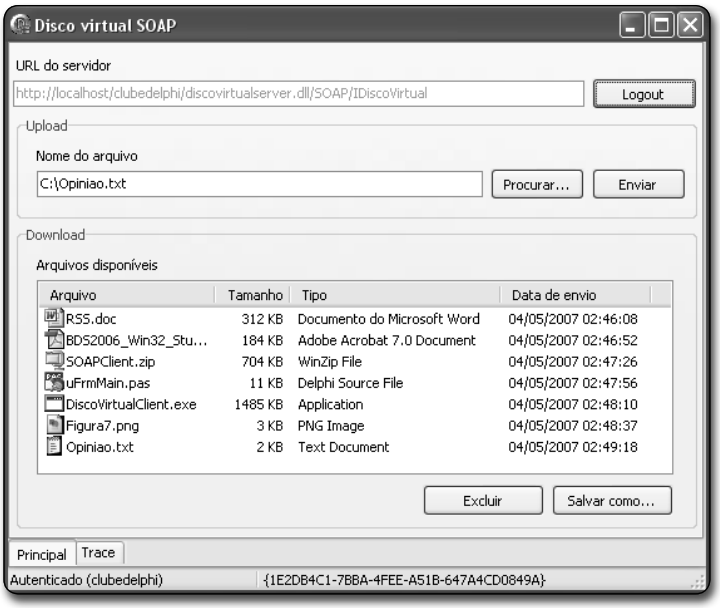


Figura 4. Formulário frmProgress



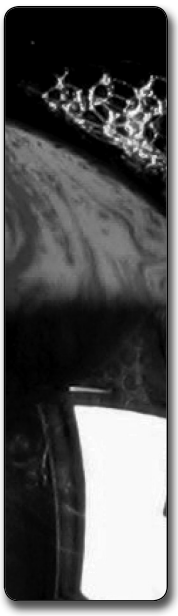


Figura 3. Arquivos remotos sendo exibidos no ListView

Listagem 11. Excluindo arquivos do banco de dados

```
procedure TDiscoVirtual.DeleteFile(const FileName: string);
var
  SessionID: string;
begin
  CheckHeader(SessionID);

  with TDataAccess.Create(nil) do
  try
    cdsFileSelect.Params.ParamByName(
      'ID_USUARIO').AsInteger := Sessions.GetUserID(SessionID);
    cdsFileSelect.Open;

    if cdsFileSelect.Locate('NOME_ARQUIVO', FileName, [loCaseInsensitive]) then
    begin
      cdsFileSelect.Delete;
      cdsFileSelect.ApplyUpdates(0);
    end;
  finally
    Free;
  end;
end;
```

Listagem 12. Excluindo um arquivo remoto

```
procedure TfrmMain.btnDeleteFileClick(Sender: TObject);
begin
  if MessageBox(
    'Você tem certeza que deseja excluir o arquivo '+'selecionado?', MB_YESNO +
    MB_ICONQUESTION) = IDYES then
  begin
    PrepareHeader;
    AppServer.DeleteFile(lstRemoteFiles.Selected.Caption);
    lstRemoteFiles.Selected.Delete;
  end;
end;
```

Listagem 13. Mostrando uma barra de progresso

```
procedure TfrmMain.ShowProgressDialog;
begin
  Self.Enabled := False;
  frmProgress.ProgressBar.Position := 0;
  frmProgress.Show;
  Application.ProcessMessages;
end;

procedure TfrmMain.HideProgressDialog;
begin
  Self.Enabled := True;
  frmProgress.Hide;
end;
```

e pressione Ctrl+Enter) e localize o *Receive*. Adicione a ele as seguintes variáveis:

```
TotalSize, TotalRead: DWORD;
```

> **Nota:** Faça uma cópia dessa unit antes de realizar qualquer alteração.

Em seguida, desça algumas linhas e localize o comentário *Extract MimeBoundary*. Adicione então **acima** dele o código a seguir:

```
Index := 0;
HttpQueryInfo(Pointer(Context),
  HTTP_QUERY_CONTENT_LENGTH or
    HTTP_QUERY_FLAG_NUMBER,
    @TotalSize, Len, Index);
TotalRead := 0;
```

Por último, localize o *repeat*, mais abaixo, e substitua as linhas:

```
if Assigned(FOnReceivingData) then
  FOnReceivingData(Size, Downloaded)
```

Por essas:

```
if Assigned(FOnReceivingData) then
begin
  Inc(TotalRead, Downloaded);
  FOnReceivingData(TotalRead, TotalSize)
end;
```

Salve tudo e adicione a pasta da unit *SoapHTTPTrans* ao *Search Path* do seu projeto ou ao *Library Path* do IDE.

> **Dica:** Para entender melhor essas alterações no código do SOAP, consulte a seção *Links*.

Comprimindo os pacotes SOAP transmitidos

Os dados transmitidos através do protocolo SOAP passam por processos de serialização e desserialização. Em nossa aplicação, os arquivos binários são expandidos

Listagem 14. Exibindo o progresso de envio e recebimento

```
procedure TfrmMain.HTTPRIOExHTTPWebNode1PostingData(Sent, Total: Integer);
begin
  frmProgress.StatusLabel.Caption := Format(
    'Enviando %d de %d kbytes', [Sent div 1024, Total div 1024]);
  frmProgress.Progressbar.Position := Trunc(Sent/Total * 100);
  if Sent = Total then
    frmProgress.StatusLabel.Caption := 'Salvando arquivo no banco de dados...';
  Application.ProcessMessages;
end;

procedure TfrmMain.HTTPRIOExHTTPWebNode1ReceivingData(Read, Total: Integer);
begin
  frmProgress.StatusLabel.Caption := Format(
    'Recebendo %d de %d kbytes', [Read div 1024, Total div 1024]);
  frmProgress.Progressbar.Position := Trunc(Read/Total * 100);
  if Read = Total then
    frmProgress.StatusLabel.Caption := 'Salvando o arquivo no disco...';
  Application.ProcessMessages;
end;
```

Listagem 15. Mostrando uma barra de progresso na aplicação

```
(...)
begin
  ...
  ShowProgressDialog; { Adicione essa linha }
  finally
   ...
    HideProgressDialog; { E essa também }
  end;
end;
```

durante esses processos, ocupando mais espaço do que o seu tamanho original.

Para melhorar a performance, podemos comprimir os pacotes enviados, de forma a diminuir o tempo necessário para a transmissão ser concluída. Isso é extremamente simples, bastando seguir os seguintes passos:

1. Compactar os pacotes no cliente antes de enviá-los ao servidor;
2. Descompactá-los no servidor antes de processá-los;
3. Compactá-los no servidor antes de enviar a resposta;
4. Descompactá-los no cliente antes de ler a resposta.

Vamos então implementar tudo isso. Já sabemos como acessar os pacotes enviados e recebidos no cliente, pois o fizemos quando criamos o *trace* das mensagens. Volte aos eventos *OnBeforeExecuteStream* e *OnAfterExecute* e modifique-os para o código da **Listagem 16**.

A classe *CompressionEngine* também faz parte da unit *Tools*. Ela usa a biblioteca de compressão *Zlib*, que acompanha o Delphi. Para entender como os métodos *CompressStream* e *DecompressStream* funcionam, não deixe de olhar esse arquivo. As chamadas ao *LogMessage* também foram alteradas, para receber corretamente agora o tamanho do pacote compactado.

Compile a aplicação e alterne para o projeto do servidor. Abra a *uWebModule*, selecione o *HTTPSoapPascalInvoker* existente e implemente seus eventos *AfterDispatchEvent* e *BeforeDispatchEvent* da forma inversa feita no cliente, isso é, descompactando **antes** e compactando **depois**.

Em seguida, descarregue a DLL do servidor, recompile o projeto e execute a aplicação cliente. Realize o envio e o recebimento de alguns arquivos e alterne para a aba *Trace* (**Figura 6**).

Observe na **Figura 12** que conseguimos uma taxa de compressão de mais de 50% na maioria dos métodos, e quase 55% no download de um arquivo!

Criptografia de pacotes

Os dados trafegados entre o cliente e o servidor são transmitidos em texto puro, sem nenhuma proteção. Se você observou o *trace* para o método *Login*, deve ter notado que a senha do usuário está lá,

Listagem 16. Comprimindo os pacotes enviados e recebidos no cliente

```
procedure TfrmMain.HTTPRIOExBeforeExecuteStream(const MethodName: string; SOAPRequest: TStream);
var
  TempStream, CompressedStream: TMemoryStream;
  OriginalSize, CompressedSize: Integer;
begin
  TempStream := TMemoryStream.Create;
  CompressedStream := TMemoryStream.Create;

  try
    SOAPRequest.Position := 0;
    TempStream.CopyFrom(SOAPRequest, 0);
    CompressionEngine.CompressStream(SOAPRequest, CompressedStream);
    OriginalSize := SOAPRequest.Size;
    CompressedSize := CompressedStream.Size;
    SOAPRequest.Position := 0;
    SOAPRequest.CopyFrom(CompressedStream, 0);
    SOAPRequest.Size := CompressedStream.Size;

    LogMessages(True, MethodName, TempStream, OriginalSize, CompressedSize);
  finally
    TempStream.Free;
    CompressedStream.Free;
  end;
end;

procedure TfrmMain.HTTPRIOExAfterExecute(const MethodName: string; SOAPResponse: TStream);
var
  TempStream, DecompressedStream,
    DecryptedStream: TMemoryStream;
  OriginalSize, CompressedSize: Integer;
begin
  DecompressedStream := TMemoryStream.Create;
  TempStream := TMemoryStream.Create;
  try
    CompressionEngine.DecompressStream(SOAPResponse, DecompressedStream);
    OriginalSize := SOAPResponse.Size;
    CompressedSize := DecompressedStream.Size;
    SOAPResponse.Position := 0;
    SOAPResponse.CopyFrom(DecompressedStream, 0);
    SOAPResponse.Size := DecompressedStream.Size;
    SOAPResponse.Position := 0;
    TempStream.CopyFrom(SOAPResponse, 0);

    LogMessages(False, MethodName, TempStream, CompressedSize, OriginalSize);
  finally
    TempStream.Free;
    DecryptedStream.Free;
  end;
end;
```

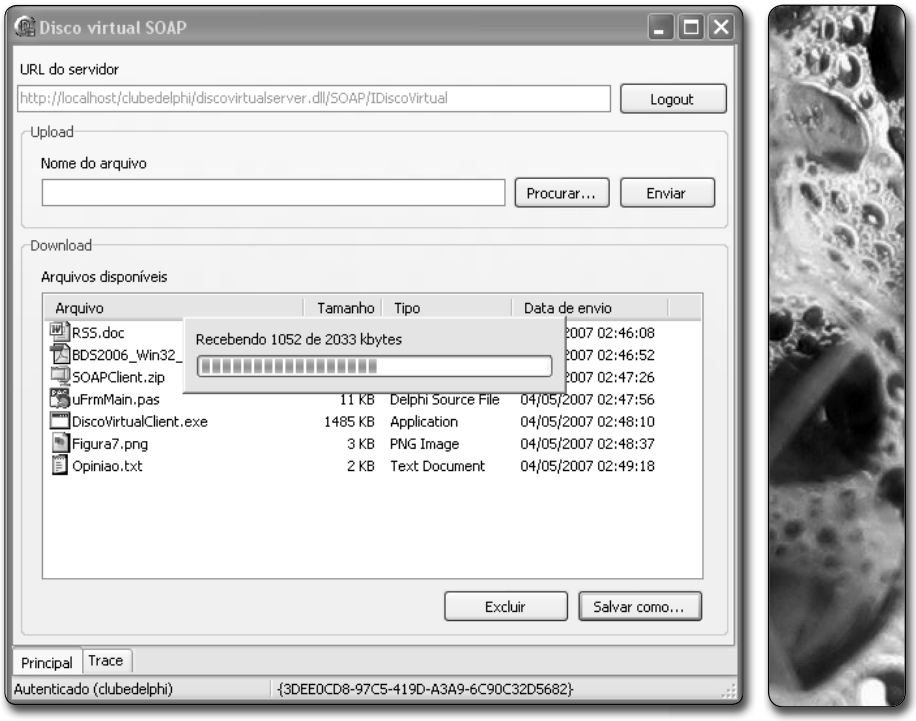


Figura 5. Barra de progresso em ação

totalmente desprotegida.

Há muitas formas de se criptografar os dados enviados e recebidos. Uma delas é através de um certificado digital, utilizando-se o protocolo HTTPS ao invés do HTTP. Outra, criptografando os pacotes antes de enviá-los, como fizemos na compressão.

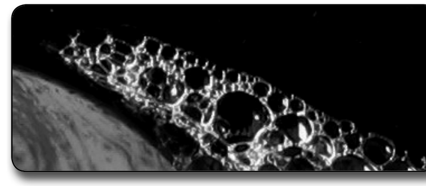
No código da aplicação disponível para download, você encontrará uma maneira de fazer isso. Eu utilizei a biblioteca de criptografia *LockBox*, da extinta *Turbo Power*. Ela é *open source* e você pode baixá-la no site do *SourceForge* (veja seção *Links*).

Nota: Tenha em mente que criptografar todos os pacotes pode comprometer a eficácia da compressão de dados, anulando-a em alguns casos.

Conclusão

Esse artigo foi extenso, mas o assunto assim o exigiu. Espero ter conseguido passar para o leitor algumas técnicas não tão óbvias sobre os Web Services criados com o Delphi, e que podem ajudar no processo de desenvolvimento de suas aplicações.

Se você tiver qualquer dúvida, sinta-se à vontade para me contatar via e-mail. Um forte abraço! ●



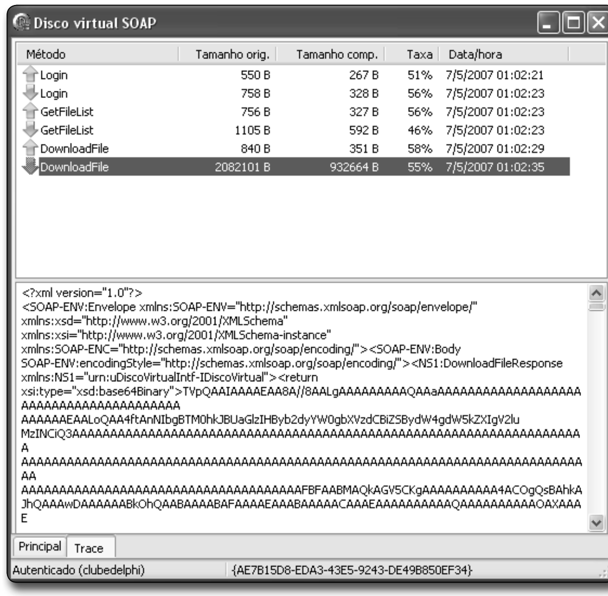
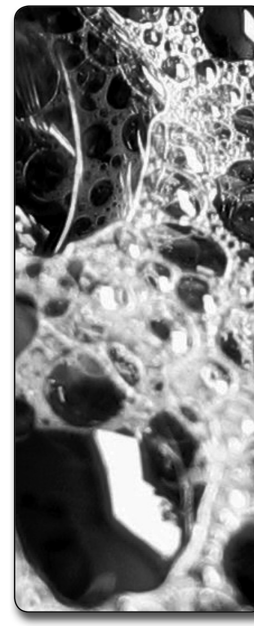


Figura 6. Comprimindo os pacotes enviados e recebidos



Impressão matricial no Delphi

Rodrigo Otto Mostaert
(otto3d@gmail.com)
é Analista/Desenvolvedor Delphi/C#/C++. Trabalha com desenvolvimentos Win32/.NET com banco de dados MySQL e Firebird em projetos Cliente/Servidor/n-tier. Especializado em sistemas WMS e possui diversos cursos oficiais Borland pela Facilit em Recife (www.facilit.com.br).

Neste artigo veremos como é simples, fácil e rápido montar um projeto que imprima dados em uma impressora matricial usando as classes do componente VDO, que é *freeware* e *open-source*. O componente VDO tanto imprime em impressoras matriciais como em impressoras de Jato.

Em nosso exemplo mostraremos como imprimir em uma impressora matricial usando a classe *VDODmPrinter*, para impressoras Jato basta usar a classe *VDOCaPrinter*. Como o VDO não envia os dados direto à porta, envia para o *Spool* do Windows, não teremos problemas com impressoras USB ou em Rede.

Criando um projeto e usando as classes

Primeiro temos que baixar o componente no site indicado na seção *Links*, poderemos instalar o componente ou adicionar os fontes dele ao projeto e instanciar as classes na mão, faremos pela segunda alternativa. Descompacte o arquivo baixado, crie um novo projeto em Delphi, copie todos os arquivos da pasta *src* do arquivo baixado e adicione-os ao projeto.

Alguns formulários automaticamente ficarão no *Auto-create* do projeto, transfira-os colocando no *Available forms*, são eles: *frmVDOPreview*, *frmVDOPrinterTypeDlg* e *frmVDOPrintProgress*. O único formulário que ficará na criação automática é o formulário principal (**Figura 1**).

Propriedades

A seguir temos as principais propriedades do VDO.

- *CurrentLine: Integer* - Leitura do número da linha atual de impressão dentro da página;
- *CurrentPage: Integer* - Leitura do número da página atual de impressão;
- *IsPrinting: Boolean* - Leitura do estado da impressora. Retorna *True* se a impressora estiver imprimindo ou *False* se não

estiver imprimindo;

• *PrintersList: TStrings* - Leitura das impressoras instaladas do Windows;

• *CharMode: TVDODmCharMode = (cmClear, cmNormal)* - Determina se será removida a acentuação do texto a ser impresso;

• *Font: TVDODmFont* - Determina a fonte a ser utilizada para impressão;

• *LineSpacing: TVDODmLineSpacing = (ls6lpp, ls8lpp)* - Determina o espaçamento vertical (entre linhas) da impressora. Se 6 ou 8 linhas por polegada;

• *Paper: TVDODmPaper* - Determina as configurações de papel e página utilizadas pela impressora;

• *PrinterName: String* - Fixa o nome da impressora a ser utilizada. Se o nome da impressora não for válido, o diálogo de impressão para seleção de uma impressora válida será exibido;

• *ShowPreview: Boolean* - Determina se o texto a ser impresso será renderizado na tela para pré-visualização da impressão;

• *ShowDialog: Boolean* - Determina se será exibido o diálogo de impressão, permitindo a seleção da impressora, número de cópias etc. Se *ShowDialog* for *True*, a fixação do nome em *PrinterName* é desconsiderada e será sobreposta pela impressora selecionada no diálogo de impressão, porém se *PrinterName* for uma impressora válida, ela estará pré-selecionada no diálogo de impressão;

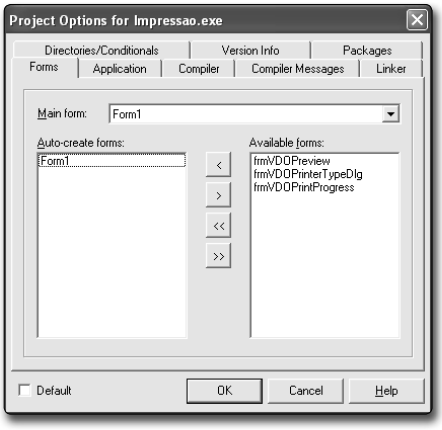


Figura 1. Definindo a criação de formulários

Figura 2. Tela do sistema

• *ShowProgress: Boolean* - Determina se será exibido o diálogo de progresso durante a impressão;

• *Title: String* - Título do trabalho de impressão a ser utilizado pelo *spool* do Windows;

Eventos

Os principais eventos são:

• *OnAbort: TNotifyEvent (VCL)* - Disparado ao abortar a impressão;

• *OnError: TNotifyEvent (VCL)* - Disparado quando ocorrer um erro na inicialização ou finalização da impressora ou durante a impressão;

• *OnNewLine: TNotifyEvent (VCL)* - Disparado a cada nova linha;

• *OnNewPage: TNotifyEvent (VCL)* - Disparado a cada nova página;

• *OnPrint: TNotifyEvent (VCL)* - Disparado a cada vez que algo é impresso;

• *OnStart: TNotifyEvent (VCL)* - Dispara-

Listagem 1. Gerando o relatório via código-fonte sem acesso ao banco

```
unit Unit1;
...
type
  tLado = (lEsquerdo, lDireito);
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
...
uses VDODmPrinter;
{ ALT+F11, selecione o VDODmPrinter e clique em OK }
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
  { Variável da classe TVDODmPrinter que fará o relatório }
  VDOImpressao: TVDODmPrinter;
begin
  VDOImpressao := TVDODmPrinter.Create(Self);
  with VDOImpressao do
  begin
    { Colocando um título ao relatório }
    Title := 'Impressão matricial DevMedia (ClubeDelphi)';
    { Informando que haverá quebra de página }
    Paper.AutoNewPage := True;
    { Informando que desejamos mostrar uma tela de preview do relatório }
    ShowPreview := True;
    { Desejamos mostrar o progresso de montagem do relatório }
    ShowProgress := True;
    { Iniciando o relatório }
    BeginDoc;
    { o Primeiro parâmetro indica a posição da coluna que iremos imprimir, o Segundo é o
      texto a ser impresso e o Terceiro, se for true o relatório vai para a próxima linha,
      caso contrário continua até chegar ao máximo da coluna informada }
    Print(00, 'Primeira Linha do relatório na posição zero', True);
    Print(10, 'Segunda Linha do relatório na posição dez',  True);
    Print(20, 'Terceira Linha do relatório na posição vinte', True);
    Print(30, 'Quarta Linha do relatório na posição trinta', True);
    NewLine(5); { Pula 5 linhas }
    Print(00, 'Linha de número.: '+Format('%3.3d', [VDOImpressao.CurrentLine]), True);
    Print(00, 'Número da página: '+Format('%3.3d', [VDOImpressao.CurrentPage]), True);
    NewPage(1); { Próxima página }
    Print(00, 'Número da página: '+Format('%3.3d', [VDOImpressao.CurrentPage]), True);
    NewPage;
    Font.Size := fsDefault;
    Font.Align := faLeft;
    Print(00, 'Texto com 10cpp e alinhado a esquerda', True);
    Font.Size := fsCondensed;
    Font.Align := faCenter;
    Print(00, 'Texto com 10cpp e alinhado ao centro', True);
    Font.Size := fsLargeCondensed;
    Font.Align := faRight;
    Print(00, 'Texto condensado expandido, c/10cpp e '+'alinhado a direita', True);
    Font.Size := fsLarge;
    Font.Align := faLeft;
    Print(00, 'Texto expandido, c/10cpp e al. '+'esquerda', True);
    Font.Size := fsDefault;
    Font.Align := faLeft;
    Font.Style := [fsDmBold];
    Print(00, 'Texto com 10cpp em "negrito"', True);
    Font.Style := [fsDmItalic];
    Print(00, 'Texto com 10cpp em "itálico"', True);
    Font.Style := [fsDmUnderline];
    Print(00, 'Texto com 10cpp em "sublinhado"', True);
    Font.Style := [fsDmBold, fsDmItalic];
    Print(00, 'Texto com 10cpp em "negrito" E '+'"italico"', True);
    { Finalizando o relatório, caso o ShowPreview seja True, abrirá uma tela com o preview
      do relatório }
    EndDoc;
  end;
  { Liberamos o objeto da memória }
  FreeAndNil(VDOImpressao);
end;
end.
```

do ao iniciar a impressão;

• *OnTerminate: TNotifyEvent (VCL)* - Disparado ao finalizar a impressão.

Métodos

Os métodos principais são:

• *Abort: Procedure* - Sua função é abortar o trabalho de impressão;

• *BeginDoc: Procedure* - Sua função é iniciar o trabalho de impressão;

• *EndDoc (AEjectPage: Boolean = True): Procedure* - Sua função é finalizar o trabalho de impressão. O parâmetro *EjectPage* indica se a página deve ser ejetada após a finalização. Se *AEjectPage* for *True*, a impressora será orientada a realizar um *Eject* antes de finalizar o trabalho de impressão. O valor *default* do parâmetro *AEjectPage* é *True*. O resultado esperado de *AEjectPage := True* depende diretamente do ajuste correto da propriedade *Lines* de **TVDODmPaper**;

• *NewLine (ALines: Integer = 1): Procedure* - Sua função é avançar o número de linhas determinadas pelo parâmetro *ALines*. Caso não seja informada a quantidade linhas o valor *default* é 1;

• *NewPage (APages: Integer = 1): Procedure* - Sua função é avançar o número de páginas determinado pelo parâmetro *APages*. Caso não seja informada a quantidade de páginas o valor *default* é 1. O resultado esperado do *NewPage* depende diretamente do ajuste correto da propriedade *Lines* de *TVDODmPaper*.

• *Print (AColumn: Integer; AText: String; ALine: Boolean = False): Procedure* - Sua função é efetuar a impressão na coluna determinada pelo parâmetro *AColumn* do texto determinado pelo parâmetro *AText*. O *ALine* é opcional e é o indicador de que a linha foi finalizada e que a impressora deve avançar para a próxima. Ao informar o parâmetro *ALine* como *True*, não é necessária a chamada posterior ao *NewLine*, a não ser que o salto de linhas adicionais seja desejado.

Colocando a mão na massa

Já temos nosso projeto pronto para montar um relatório para imprimir. O VDO dá uma boa facilidade na hora de programar, devido a sua simples codificação e um resultado bastante satisfatório na hora de imprimir. Chega de conversa e vamos ao que interessa. No formulário principal coloque dois botões (**Figura 2**).

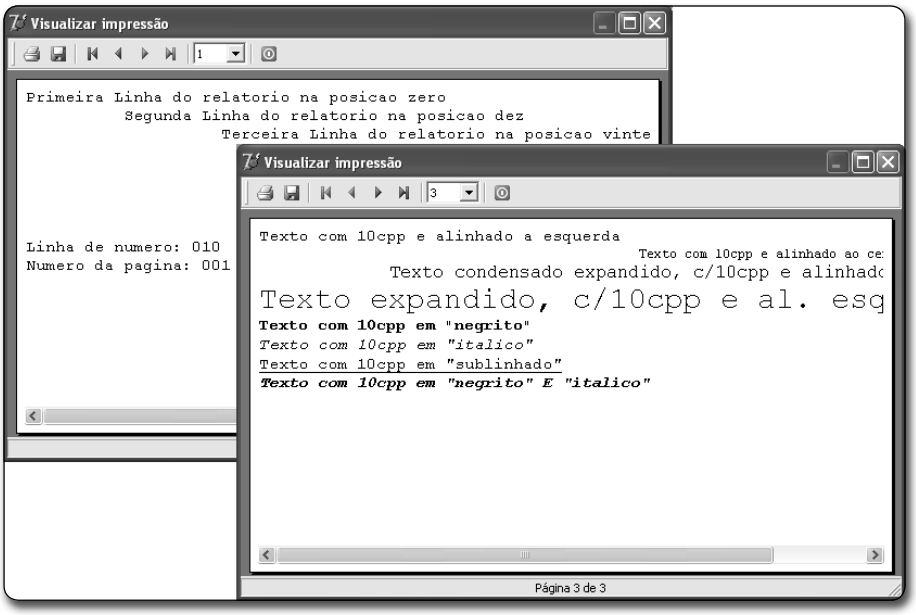


Figura 3. Preview do relatório usando o VDO

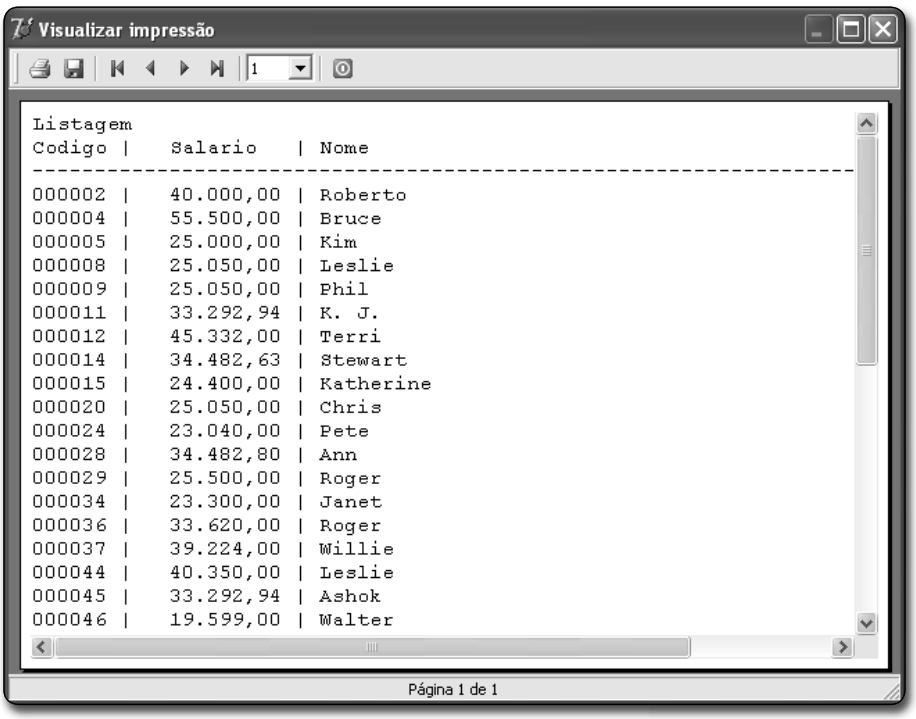


Figura 4. Preview do relatório usando o VDO

No evento *OnClick* do *Gerar relatório sem banco* faremos um exemplo sem acesso a banco, com o código da **Listagem 1**.

O código da listagem anterior adiciona várias linhas ao objeto de impressão com várias formas de formatação, como: tipo da fonte, alinhamento, tamanho, estilo da letra etc. A **Figura 3** mostra o resultado final do nosso exemplo sem acesso a um banco de dados.

Mostrando dados do banco

No evento *OnClick* do *Gerar relatório com banco* faremos um exemplo acessando o banco de dados do BDE (tabela *Employee*) com o código da **Listagem 2**. Você pode adaptar o exemplo para qualquer tecnologia de acesso a dados e banco, como o Firebird ou SQL Server.

Listagem 2. Gerando o relatório com BD

```
uses DBTables;
...
procedure TForm1.Button2Click(Sender: TObject);
  function CompletaComBranco(AValue: string; AQuantidade:integer; ALado: tLado): string;
  var
    contador: integer;
  begin
    Result := '';
    for contador := 1 to Abs(AQuantidade - Length(
      Avalue)) do
    begin
      Result := Result + ' ';
    end;
    if ALado = lEsquerdo then
      Result := Result + AValue
    else
      Result := AValue + Result;
  end;

var
  { o Qry fornecerá os dados do banco }
  Qry: TQuery;
  { Variável da classe TVDODmPrinter que fará o relatório }
  VDOImpressao: TVDODmPrinter;
  Linha: string;
begin
  Qry := TQuery.Create(Self);
  { Informa o alias do banco }
  Qry.DatabaseName := 'DBDemos';
  { Instrução SQL }
  Qry.SQL.Text :=
    'select empno, firstname, salary from employee';
  { Abre o DataSet pra que traga os registros da tabela }
  Qry.Open;
  VDOImpressao := TVDODmPrinter.Create(Self);
  VDOImpressao.Title := 'Impressão matricial DevMedia (ClubeDelphi)';
  VDOImpressao.Paper.AutoNewPage := True;
  VDOImpressao.ShowPreview := True;
  VDOImpressao.ShowProgress := True;

  with VDOImpressao do
  begin
    BeginDoc;
    Print(0, 'Listagem', True);
    Print(0, 'Código |   Salário   | Nome', True);
    Print(0, StringOfChar('-',80), True);
    { Posiciona no primeiro registro }
    Qry.First;

    { Enquanto não chegar no último registro imprime no relatório }
    while not Qry.Eof do
    begin
      { Na variável Linha será feito a concatenação
        dos dados pra que saiam formatados }
      Linha := Format('%6.6d',
        [Qry.FieldByName('empno').AsInteger])+' | '+
        CompletaComBranco(FormatFloat('#,###,##0.00', Qry.FieldByName('salary').AsFloat), 11,
        lEsquerdo)+' | '+ Qry.FieldByName('firstname').AsString;
      Print(0, Linha, True);
      { Próximo registro }
      Qry.Next;
    end;
    FreeAndNil(Qry);
    EndDoc;
  end;
  FreeAndNil(VDOImpressao);
end;
```

O código da listagem anterior cria uma conexão com o banco *dbdemos* e faz uma consulta na tabela *employee* trazendo todos os dados. Criamos um laço e adicionamos os registros dentro do objeto com uma formatação simples de cada linha um registro. A **Figura 4** mostra o nosso exemplo acessando um banco de dados.

Conclusão

O exemplo desenvolvido é bem básico, tendo como objetivo mostrar como é simples, rápido e fácil fazer um relatório para imprimir em impressora matricial em Delphi usando os componentes VDO.

Recomendo o uso desses componentes, pois, é realmente muito rápido a impressão usando as classes do VDO, basta pesquisar as propriedades e eventos a fundo do mesmo e olhar os exemplos em anexo ao componente.

Bons estudos e até a próxima! ●

Instalando os componentes no IDE

Como alternativa a instanciar os componentes na mão como fizemos, é possível fazer a instalação dos componentes do VDO no IDE do Delphi. Ao fazer o download do arquivo, descompacte-o na pasta desejada. No IDE do Delphi acesse File>Open, e na pasta de instalação, localize a sub-pasta packages.

Acesse a sub-pasta correspondente a sua versão do Delphi e abra o arquivo VDOPrint.dpk. Clique em Compile e depois em Install. Feche o pacote (não há necessidade de salvar). No IDE do Delphi vá ao menu Tools>Enviroment Options e na aba Library acesse Library Path.

Adicione pasta de instalação/src do componente e feche os editores. Veja na **Figura 5** os componentes instalados no IDE do Delphi.

A seguir, temos uma descrição de cada componente:
- TVDOPrinter: Destinado a impressão em impressoras jato de tinta, laser ou matriciais. É uma classe que encapsula as propriedades, eventos e métodos das classes TVDOCaPrinter e TVDODmPrinter, com apenas uma funcionalidade a mais que é permitir ao desenvolvedor e/ou ao usuário (opcionalmente) a escolha do tipo de impressora desejada no momento do início da impressão;

- TVDOCaPrinter: Destinado a impressão em impressoras jato de tinta e laser. Utiliza internamente o objeto TPrinter da VCL em conjunto com o Canvas para efetuar a escrita. Utiliza o driver instalado no Windows e obedece a todas as configurações de perfil do usuário e da impressora. Pode ser utilizado em impressoras matriciais, mas a impressão é consideravelmente mais lenta;

- TVDODmPrinter: Destinado a impressão em impressoras matriciais. Efetua gravação do documento no spool de impressão do Windows. Não envia os dados diretamente às portas de impressão, e por esse motivo pode ser usado, sem nenhum problema, com impressoras USB e de rede. Utiliza o padrão Epson de caracteres de controle da impressora (Escapes).

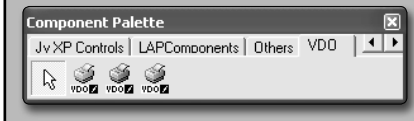


Figura 5. Componentes instalados no IDE

Links

Site do VDO
vdo.sourceforge.net

Publique na Web seus relatórios do QuickReport

Fabrício Desbessel
(fabricio@fabricio.pro.br)
é professor de Linguagem de Programação do Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração está sempre disposto a provar que com o Delphi sempre teremos a melhor solução. Site www.fabricio.pro.br.

Este artigo vai mostrar como é possível aproveitar relatórios do QuickReport para serem exibidos na Web, através da exportação para PDF. Com isso, você não precisará reconstruir seus relatórios existentes para que possam ser disponibilizados na Web, podendo ser acessados a partir de qualquer lugar da Internet.

Instalações necessárias

Primeiramente precisamos ter a paleta de componentes do *QuickReport* disponível no IDE do Delphi 7. Se você não está com a paleta visível, acesse o menu *Component>Install Packages*. Clique no botão *Add* e, dentro do diretório *Bin* do Delphi, escolha o arquivo *dclqrt70.bpl*.

Como o *QuickReport* não possui um componente ou método para exportar o relatório em PDF, será necessário utilizar o componente *ExportQR*. A instalação é simples (o componente está para download com os fontes do artigo).

Criando o projeto Web

Como a idéia é utilizar relatórios já desenvolvidos, vamos adicionar as units que contenham relatórios ao nosso projeto. Também teremos uma página inicial, que funcionará como um menu, exibindo todos os relatórios disponíveis ao usuário.

Para criar uma aplicação *WebBroker* acesse o menu *File>New>Other* e na aba *New* escolha o ícone *Web Server Application* (**Figura 1**).

Na tela seguinte escolha o tipo de aplicação *CGI Stand alone executable*. Se você tem interesse em saber as diferenças desses tipos de aplicação, pode acessar um artigo de minha autoria, através do link: *www.devmedia.com.br/articles/viewcomp.asp?comp=1652*. Salve a unit e para o pro-

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp
Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta que mostra como trabalhar com o ExportQR.
www.devmedia.com.br/articles/viewcomp.asp?comp=3130

jeto informe o nome "Relatorios".

Nesse momento, temos disponível um *Web Module* (parecido com o Data Module) para adicionar os componentes necessários para o funcionamento da aplicação, bem como as ações e códigos.

Adicionando os relatórios

Nesse momento já é possível adicionar ao projeto as units dos relatórios existentes. Para tanto, clique no botão *Add file to project* e selecione as units que contenham relatórios, desenvolvidos com o *QuickReport*.

Depois de adicionar as units ao projeto, é necessário certificar-se que a aplicação não criará os formulários na sua inicialização, o que poderá gerar uma exceção. No menu *Project>Options>Forms* coloque todos os formulários de relatórios para *Available forms* (direita), deixando somente o *WebModule1* no *Auto-create forms* (esquerda).

> **Nota:** É importante salientar que neste exemplo, os formulários que contenham os relatórios do *QuickReport* possuam os componentes de acesso a dados no próprio relatório.

Nesse momento precisamos dizer ao *WebModule1* para utilizar as units dos formulários que contenham relatórios. Para fazer isso acesse o menu *File>Use Unit*, selecione os formulários e pressione OK.

Para que o *WebModule* saiba quais os relatórios estão disponíveis, vamos criar *arrays* para os formulários, classes, opções de menu, bem como variáveis para a quantidade de relatórios e link dos PDFs gerados. Acesse o código do *WebModule1* e na seção *var* acrescente o código da **Listagem 1**.

Depois de declarar as variáveis devemos informar os formulários que foram adicionados ao projeto. Isso é feito no evento *OnCreate* do *WebModule1*. Para exemplificar, codifiquei para dois relatórios, conforme mostra a **Listagem 2**.

Para cada relatório você deverá informar sua classe, seu *Name* e o nome que aparecerá no menu. Ao final, a variável *Qtde* deve receber a quantidade de relatórios inseridos no projeto e que estarão disponíveis na Web.

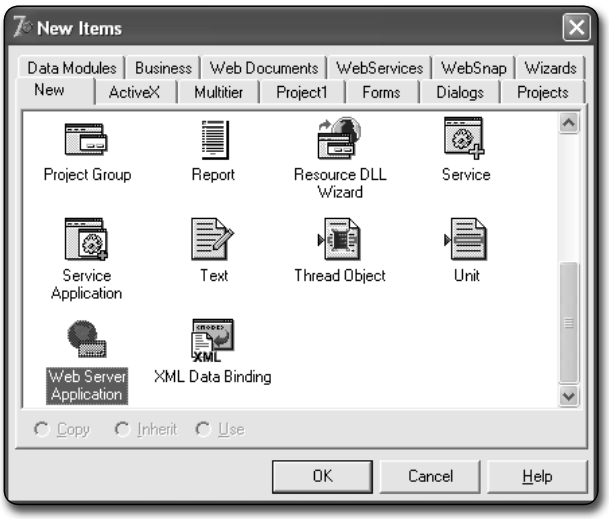


Figura 1. Criando o projeto Web Server Application

Listagem 1. Declarando variáveis ao projeto

```
uses Forms;
...
Classes: Array[1..5] of TFormClass;
Formularios: Array[1..5] of TForm;
Menu: Array[1..5] of string;
Qtde: Integer;
Link: string;
```

Listagem 2. Informando os relatórios disponíveis no OnCreate

```
procedure TWebModule1.WebModuleCreate(Sender: TObject);
begin
  Classes[1] := TrelEmpregados;
  Formularios[1] := relEmpregados;
  Menu[1] := 'Listagem de Empregados';
  Classes[2] := TrelPaises;
  Formularios[2] := relPaises;
  Menu[2] := 'Listagem de Países';
  Qtde := 2;
end;
```

Listagem 3. Código HTML do PageProducer ppMenu

```
<html>
<head><title>Clube Delphi - Exportação PDF</title></head>
<body>
<p><font color="#000000"><strong>
  <font size="3" face="Verdana, Arial, Helvetica, sans-serif">
     Sistema Relat&oacute;rios na WEB - Clube Delphi</font>
  </strong></font></p><p>
  <font size="2"><strong><font color="#000000" face="Verdana, Arial, Helvetica, sans-serif">
     Menu de Relat&oacute;rios</font>
  </strong></font></p>
<hr/>
<p><#menu></p>
</body>
</html>
```

Criando as ações

Primeiramente vamos criar uma ação (*action*), que exibirá uma página listando os relatórios disponíveis. Clique duas vezes sobre o *WebModule* para abrir o editor de ações. Clique no botão *Add New* para inserir.

Altere a propriedade *Name* para "waiMenu", coloque "/Menu" na propriedade *PathInfo* e mude para *True* a propriedade *Default*. Antes de colocar o código da ação precisamos adicionar mais um componente ao *WebModule*.

Da paleta *Internet*, adicione um *PageProducer* e altere seu *Name* para "ppMenu". Na propriedade *HTMLDoc* informe o código da **Listagem 3**.

Note que no código, que é um HTML normal, temos uma tag *<#menu>* que o Delphi reconhecerá e codificaremos para, no lugar dela, ser exibido a lista de relatórios. Para isso acesse o evento *OnHTMLTag* do *ppMenu* e coloque o código da **Listagem 4**.

Volte ao editor de ações, clicando duas vezes sobre o *WebModule* e selecione a ação *waiMenu*. Acesse o evento *OnAction* e adicione o seguinte código:

```
Response.Content := ppMenu.Content;
```

Nesse momento podemos fazer nosso primeiro teste. Lembre-se antes de compilar, de mudar o diretório de saída do projeto para ficar dentro de *C:\Inetpub\Scripts* que é o diretório padrão do IIS que executa scripts CGI.

> **Importante:** É necessário dar permissão de execução ao diretório *Scripts* para que ocorra o funcionamento correto da aplicação, além de ter o IIS instalado claro. Para isso, acesse o *Adicionar/Remover Programas* no Painel de Controle. Clique em *Adicionar/remover componentes do Windows*, escolha *Internet Information Services (IIS)* e clique em *Detalhes*. Na janela seguinte, escolha *Serviço da World Wide Web* e clique em *Detalhes*. Marque a opção *Diretório Virtual Scripts* e clique em OK nas janelas abertas.

Para fazer essa mudança acesse o menu *Project>Options*, aba *Directories/Conditionals* e *Output directory*. Compile a aplicação com o CTRL+F9 e abra no *browser* o seguinte endereço:

```
http://localhost/scripts/Relatorios.exe
```

> **Nota**: Se você tiver um programa para acelerar *downloads*, ele pode tentar salvar o arquivo ao invés de executar. Para contornar, digite juntamente o *PathInfo* da ação. Nesse caso o link ficaria: *http://localhost/scripts/Relatorios.exe/Menu*.

Será mostrada uma tela semelhante a **Figura 2**.

Exportando para PDF

Para exportar os relatórios escolhido no menu, crie uma nova ação no *WebModule* (clicando duas vezes sobre o mesmo) e depois no *Add New*. Configure a propriedade *Name* para "waiRelatorio" e *PathInfo* para "/Rel".

Adicione ao *WebModule* um *ExportQR*. Coloque também mais um *PageProducer*, alterando seu *Name* para "ppRelatorio" e informando o código da **Listagem 5** na propriedade *HTMLDoc*.

A idéia do *ppRelatorio* é redirecionar a página para o PDF exportado, através de uma função *JavaScript*. Em tempo de execução vamos alterar a tag <*#rel*> para o caminho do PDF. Para isso, no evento *OnHTMLTag* do *ppRelatorio*, adicione o seguinte código:

```
if TagString = 'rel' then
  ReplaceText := 'http://localhost/'+Link+'.PDF';
```

Na seção *private* do *WebModule*, declare a função *Exporta*, conforme o seguinte código:

```
function Exporta(Classe: TFormClass;
  Formulario: TForm): string;
```

Pressione SHIFT+CTRL+C para que o Delphi implemente a estrutura da função e codifique-a conforma é apresentado na **Listagem 6**.

A função será responsável por criar o formulário em tempo de execução, preparar o relatório e exportá-lo. Você pode exportar para qualquer diretório que

Listagem 4. OnHTMTag do ppMenu

```
procedure TWebModule1.ppMenuHTMLTag(Sender: TObject;
  Tag: TTag; const TagString: string;
  TagParams: TStrings; var ReplaceText: string);
var
  i: integer;
begin
  if TagString = 'menu' then
  begin
    for i := 1 to Qtde do
    begin
      ReplaceText := ReplaceText + '<a href="http://localhost/Scripts/' +
        'Relatorios.exe/Rel?Form=' + IntToStr(i) + '" target="_blank">' +
      Menu[I]+'</a><br>';
    end;
  end;
end;
```

Listagem 5. Código HTML do PageProducer ppRelatorio

```
<html>
<head>
<title>Clube Delphi - Exportação PDF</title>
<script language="JavaScript">
<!--
function MM_goToURL() {
  var i, args=MM_goToURL.arguments;
  document.MM_returnValue = false;
  for (i=0; i<(args.length-1); i+=2) eval(args[i]+".location='"+args[i+1]+"'");
}
//-->
</script>
</head>
<body bgcolor="#FFFFFF"
  onLoad="MM_goToURL('parent','<#rel>');
  return document.MM_returnValue">
</body>
</html>
```

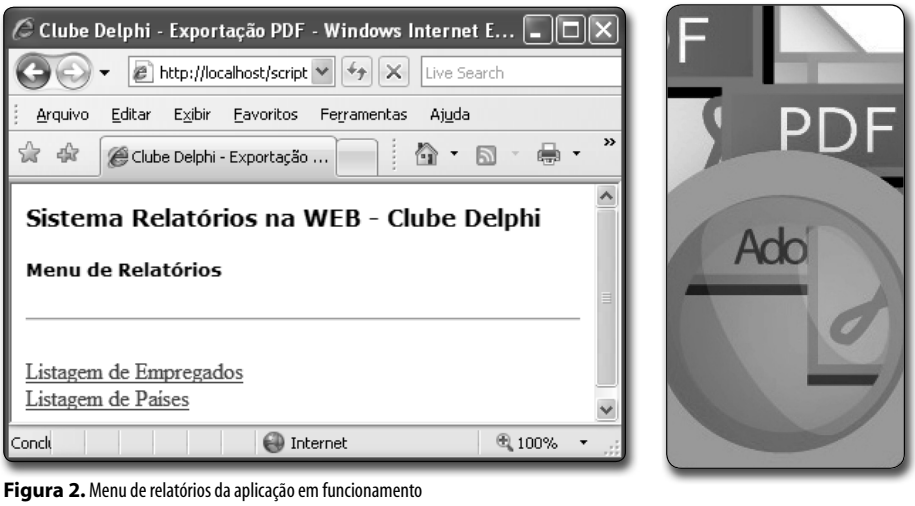


Figura 2. Menu de relatórios da aplicação em funcionamento

Listagem 6. Código da função Exporta

```
uses QuickRpt;
...
function TWebModule1.Exporta(Classe: TFormClass; Formulario: TForm): string;
begin
  try
    Formulario := Classe.Create(WebModule1);
    ExportQR1.Report:= Formulario.FindComponent('QuickRep1') as TQuickrep;
    (Formulario.FindComponent('QuickRep1') as TQuickrep).Prepare;
    ExportQR1.ExportQRPDF('C:\inetpub\wwwroot\'+Formulario.Name, True);
    Result := Formulario.Name;
  except
    Result := '';
  end;
end;
```

Listagem 7. OnAction da ação waiRelatorio

```
procedure TWebModule1.WebModule1waiRelatorioAction(Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
var
  Consulta: string;
  Indice: Integer;
begin
  Consulta := Request.QueryFields.Values['Form'];
  if Consulta <> '' then
  begin
    Indice := StrToInt(Consulta);
    Link := Exporta(Classes[Indice], Formularios[Indice]);
    if Link <> '' then
      Response.Content := ppRelatorio.Content
    else
      Response.Content := 'Não foi possível gerar o relatório!';
  end;
end;
```

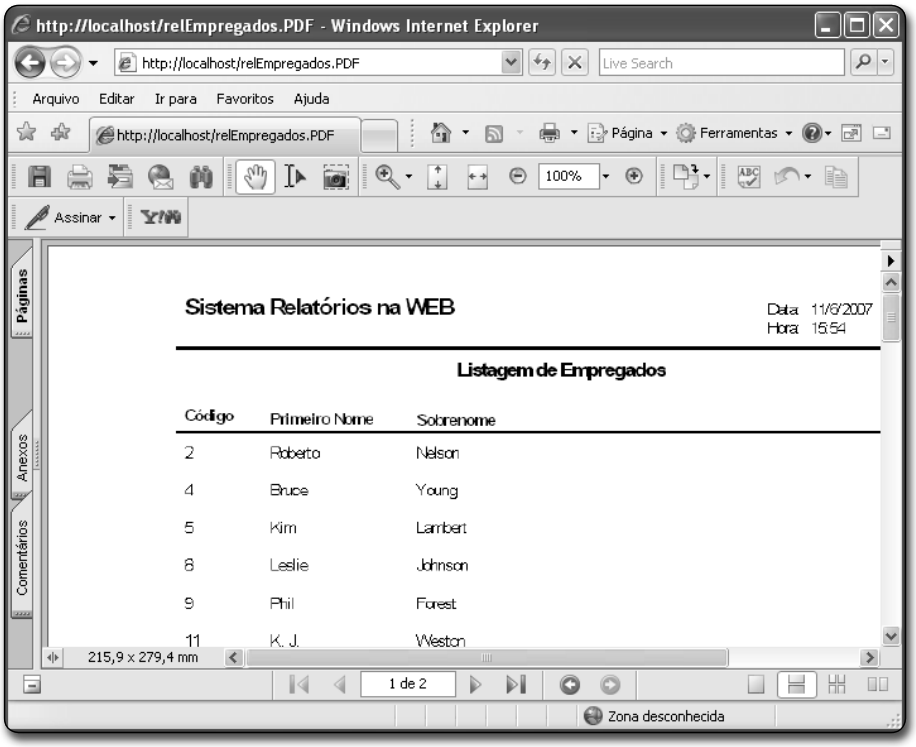


Figura 3. Relatório exportado na Web

esteja disponível no IIS. No exemplo foi exportado para *C:\inetpub\wwwroot* que é o diretório padrão de páginas do IIS.

> **Nota**: Ao publicar a aplicação em um servidor, lembre-se que no diretório onde serão exportados os PDFs, a conta IUSR deverá ter permissão para gravar os arquivos.

Para finalizar, vamos codificar a *waiRelatorio*. No evento *OnAction* coloque o código da **Listagem 7**.

Agora compile a aplicação e abra no *browser* o endereço: *http://localhost/scripts/Relatorios.exe/Menu*. No menu de relatórios, clique em link um e confira a abertura do relatório em PDF (**Figura 3**). Lembre-se que é necessário ter instalado algum software leitor de PDF na sua máquina.

Conclusão

Neste artigo vimos uma forma interessante de reutilizar relatórios desenvolvidos em *QuickReport*, disponibilizando-os para serem acessados pela Web, dando mobilidade aos usuários que poderão acessá-los de qualquer ponto de internet.

Lembre-se de criar um controle de usuários que poderão ter acesso aos relatórios e ainda podendo definir grupos de relatórios que cada usuário pode acessar. Cuidado com o número de usuários online que utilizarão o sistema. Para permitir mais conexões ao mesmo tempo opte por uma aplicação ISAPI ao invés de CGI. Pense nisso. Coloque a mão na massa e disponibilize seus relatórios na Web. ●

Boas práticas para trabalhar em equipe

Fernando Sarturi Prass
(fernando@dotbr.com.br)
é Mestre em Ciência da Computação pela UFSC. Professor da Universidade Luterana do Brasil (ULBRA) nos campus de Santa Maria e Cachoeira do Sul. Sócio-diretor da dotBR Soluções em TI (www.dotbr.com.br), empresa que presta serviços de desenvolvimento de sistemas e de consultoria em Bancos de Dados e Metodologias de Desenvolvimento.

Cristiano Caetano
(c_caetano@hotmail.com)
é certificado pela Associação Latino-Americana de Teste de Software e autor do livro CVS: Controle de Versões e Desenvolvimento Colaborativo de Software. Com 10 anos de experiência, já trabalhou na área de qualidade e teste de software para grandes empresas como Zero G, DELL e HP Invent. O autor também pode ser contatado em spaces.msn.com/softwarequality.

As metodologias de desenvolvimento foram criadas visando coordenar esforços, estabelecer ordem, padrões e técnicas. Entre as melhores metodologias estão O *Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos (PMBOK)* do *Project Manager Institute* (PMI), o *Capability Maturity Model* (CMM) e o modelo brasileiro desenvolvido pela SOFTEX baseado no CMMI chamado de *Melhoria de Processo do Software Brasileiro* (MPS.BR).

Por essas metodologias serem bastante completas, o seu uso, muitas vezes, se torna difícil em pequenas empresas ou pequenos grupos de programadores. Entretanto, por menor que seja o grupo ou a empresa, o desenvolvimento de sistemas em equipe requer uma série de cuidados para que o trabalho de um não seja refeito ou alterado erroneamente por outro membro do time.

Este artigo mostrará uma série de pequenas dicas e ferramentas, muitas delas retiradas dos modelos já citados, que visam facilitar o trabalho em grupo e, mais do que isso, garantir que o grupo trabalhe de forma correta e gere ao final do processo um produto de qualidade.

Divisão de tarefas

O fator fundamental para que o trabalho em equipe obtenha sucesso é a correta divisão das tarefas. A forma mais correta a ser seguida na divisão é tentar atender a duas afirmações: *cada um deve fazer aquilo que sabe* (assim não haverá necessidade de tempo de aprendizado) e *sempre que possível deve fazer sozinho* (a divisão de uma tarefa entre duas ou mais pessoas, muitas vezes obriga um dos membros da equipe a ficar parado esperando pela finalização do trabalho do outro).

Por exemplo, suponha um determinado sistema que usa funções que estão num Web Service. Pode-se designar um programador experiente para codificar as classes e exportá-las como Web Services e passar a construção das telas e dos mó-

dulos locais (processos bem mais simples) para programadores com menor experiência ou até mesmo para estagiários.

A divisão correta das atividades do projeto traz benefícios como a economia com programadores, pois não é necessário manter um time inteiro de profissionais experientes, e facilidades no controle de versão e na gerência da segurança (no exemplo dado, os profissionais com menor experiência não precisariam ter acesso aos fontes das classes por exemplo).

Para finalizar, é preciso que os membros da equipe saibam exatamente quem faz o que no projeto. Esse artigo traz uma seção intitulada *Matriz de Designação de Responsabilidades* que trata sobre o assunto.

Documentação da Análise

O projeto de um novo *software* começa sempre pela análise. Bem, ao menos deveria começar! É por essa razão que se diz que a documentação é a base de qualquer trabalho de desenvolvimento.

Os *softwares* possuem um ciclo de vida com constantes alterações. A documentação é um importante instrumento a ser utilizado para manter o histórico, pois conta toda a história do aplicativo, desde sua concepção, passando pelo desenvolvimento e chegando até o acompanhamento da manutenção.

Pode parecer desnecessário fazer tais afirmações já que elas são de senso comum, mas muitas vezes essa fase não é concluída em sua plenitude para que sobre mais tempo para a fase de programação.

Entretanto para que a documentação possa cumprir o seu papel, ela necessita ser elaborada durante cada uma das fases de desenvolvimento e ser atualizada sempre que houver uma modificação, independente da metodologia adotada pela empresa. Deve ser vista como uma ferramenta de auxílio a ser utilizada por todas as pessoas envolvidas no processo e não como uma obrigação a mais.

Para que o processo de documentação não se torne um entrave ao andamento do projeto é preciso que a equipe escolha corretamente que tipos de documentos serão gerados na fase de análise. Não é necessário dizer que o ideal seria gerar o máximo de documentos úteis possíveis, entretanto a realidade (leia-se: a falta de tempo) não permite que isso seja feito.

Para ganhar tempo, sem perder qualidade, os autores aconselham que a equipe faça *ao menos* os seguintes documentos:

- Estrutura Analítica do Projeto (EAP);
- Casos de uso e Diagramas de Caso de Uso (*Use Cases*);
- Diagramas de Classes;
- Modelo de Dados.

A Estrutura Analítica do Projeto (**Figura 1**), também conhecida como *Work Breakdown Structure* (WBS), é uma estrutura em forma de árvore que organiza e define o escopo total do projeto. É uma ferramenta de decomposição do trabalho, cujo objetivo principal é identificar os itens que serão desenvolvidos (PMBOK, 2004).

Aqueles que estão habituados ao uso de softwares de gestão de projetos, como o Project ou Open WorkBench não encontrarão nenhuma dificuldade para montar uma. Maiores informações sobre a EAP podem ser obtidas em *en.wikipedia.org/wiki/wbs*.

Os Diagramas de Caso de Uso e de Classes não serão explicados aqui por serem amplamente conhecidos pelos desenvolvedores, além do mais, esse não é o foco do artigo. Maiores informações sobre eles pode ser obtidos em Spínola e Araújo (2006) na seção *Links*.

Documentação do código-fonte

Embora essa seção também trate sobre documentação, ela foi separada da anterior por ser a fase de desenvolvimento mais longa do projeto, logo é nela que estão concentrados os maiores esforços da equipe.

Pode-se afirmar que a documentação do código-fonte é um dos artefatos que tornam o desenvolvimento e manutenção de sistemas, seja ele individual ou coletivo, mais simples e eficiente.

O problema é que normalmente a documentação do código não é feita ou, quando feita, não é atualizada. As razões (para não escrever as desculpas) mais comuns são: falta de tempo, indisposição ou até mesmo o desconhecimento de suas vantagens.

Por causa da complexidade e principalmente da alta mutabilidade do código-fonte, não basta apenas elaborar o documento inicial, é preciso também que ele acompanhe todas as manutenções e alterações efetuadas no sistema, para que seja mantida a integridade das

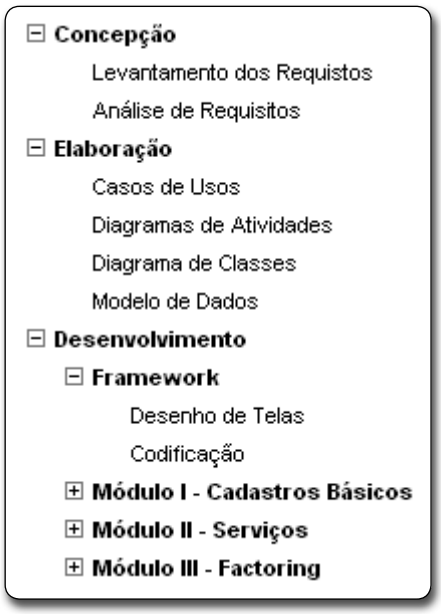


Figura 1. Parte da EAP de um projeto de desenvolvimento

informações contida nele.

Existem motivos que praticamente obrigam a documentação do código-fonte, vale destacar os seguintes:

- Evita o retrabalho (exemplo: um programador escrever uma classe ou função já desenvolvida por outro);
- Diminui o número de interrupções para sanar dúvidas (ao invés de perguntar para quem escreveu determinado método quais parâmetros que ele aceita, basta que olhe o documento);
- Mantém o histórico do projeto em todo o seu ciclo de vida;
- Facilita a substituição e a incorporação de membros à equipe de desenvolvimento;
- Facilita a visualização do tamanho do produto, uma vez que todos os módulos podem ser reunidos em um único documento;
- Acrescenta qualidade ao produto final entregue ao cliente.

Além dos motivos citados, existe um que é o principal: facilita a manutenção do sistema. No site da SoftConsult (*www.softconsult.se*) tem um provérbio que diz: ***If it was hard to write, it should be hard to read*** ou algo como se foi difícil codificar, deve ser mais difícil ler e entender o código.

Essa frase define bem a dificuldade encontrada pelos programadores no momento em que são obrigados a dar manutenção num determinado sistema. Entretanto não basta que exista um do-

cumento, é preciso que ele tenha uma formatação clara e padronizada a fim de que sua leitura seja facilitada.

Uma vez escolhido um padrão para títulos, subtítulos, corpo, comandos e comentários, ele deve ser seguido durante todo o documento. Outro ponto importante é o formato do documento, deve-se dar prioridade a um que possua fácil distribuição e recursos que facilitem a ligação entre as seções. O formato ideal é o HTML, pois é multi-plataforma e permite o uso de imagens e outros recursos.

Felizmente hoje a documentação do código-fonte pode ser feita de maneira quase automática, através de softwares específicos como, por exemplo, o DelphiDoc (**Figura 2**).

Desenvolvido pela empresa sueca SoftConsult baseado no JavaDoc da Sun (que documenta códigos Java como o próprio nome sugere), o DelphiDOC gera um site HTML com toda a documentação do sistema (classes, métodos, funções, arquivos etc.).

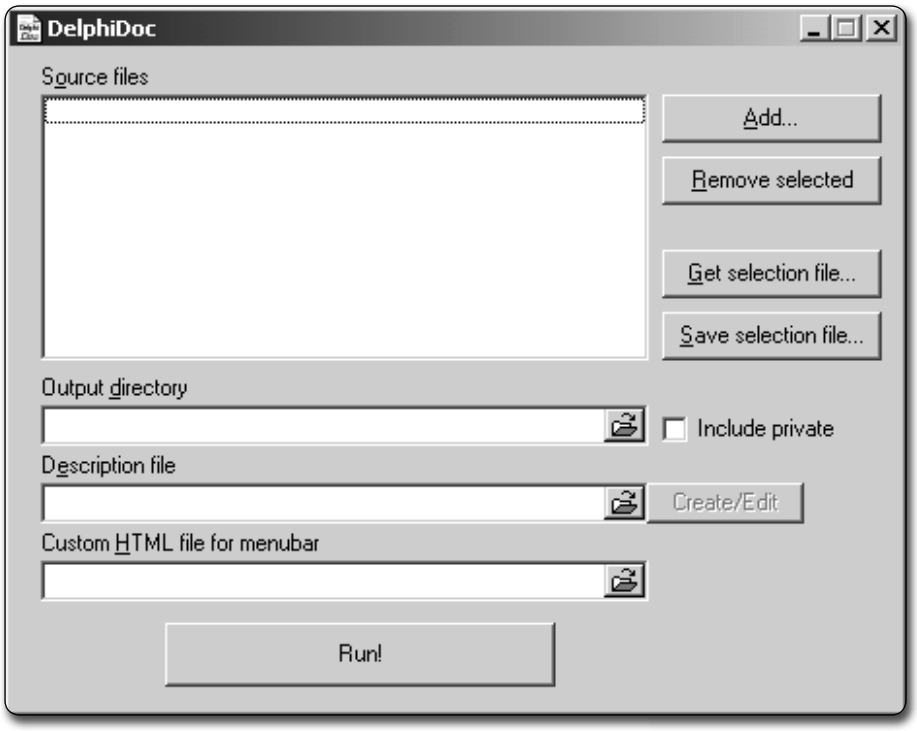


Figura 2. Tela inicial do DelphiDOC

Padrões de Projetos

Observe os dois códigos na **Listagem 1,** eles fazem exatamente a mesma coisa. Na verdade são praticamente idênticos, mas estão formatados de maneira diferente. Se num código tão pequeno a diferença já é gritante, imagine num código-fonte com duzentas linhas ou mais.

Para piorar a situação, suponha que a equipe seja formada por três pessoas, cada uma escrevendo a sintaxe de uma forma. Se, por acaso, algum programador tiver que dar manutenção em um código escrito por outro, antes mesmo de aprender as regras de negócio do módulo que necessita de manutenção, ele terá de aprender a ler o programa escrito pelo colega, aumentando assim o tempo de trabalho.

Os padrões de projetos existem, entre outras coisas, justamente para evitar isso. A **Listagem 2** traz alguns exemplos de padrões a serem seguidos por todos os programadores durante a codificação para tornar o código o mais homogêneo, facilitando assim a leitura e a manutenção do programa.

O leitor pode adaptar a sintaxe dos comandos para que fique da forma como já está acostumado (por exemplo: ao invés de usar a forma *variavel_composta* pode

Listagem 1. Exemplos de comandos

```
{ Exemplo 1 }
if variavel = 1 then begin
  meu_contador := 1;
  valor_total := 0;
  while meu_contador < 10 do begin
    valor_total := valor_total + meu_contador;
    meu_contador = meu_contador +1;
  end;
end
else valor_total := 10;

{ Exemplo 2 }
if Variavel = 1 then
begin
  MeuContador := 1;
  ValorTotal := 0;
  while MeuContador < 10 do
  begin
    ValorTotal := ValorTotal + MeuContador;
    MeuContador := MeuContador +1;
  end;
end
else
begin
  ValorTotal := 10;
end;
```

Listagem 2. Exemplo de padrões para comandos

```
variavel
variavel_composta
for <variavel> := <inicio> to <fim> do
begin
  <comandos>;
end;

if <condicao> then
begin
  <comandos>;
end

else
begin
  <comandos>
end;

(...)
```

usar *VariavelComposta*). O importante é que toda a equipe receba um documento com o formato que deve escrever o código-fonte (*if, while, for, try..except*, nome de arquivos, nome de classes etc.) e use esse padrão durante todo o projeto.

Matriz de Designação de Responsabilidades

Tão importante quanto escrever tudo da mesma forma, é saber quem é o responsável pelo que no projeto. Para tanto, existe um artefato chamado de Matriz de Designação de Responsabilidades (RAM - *Responsibility Assignment Matrix*) que define as responsabilidades de cada membro da equipe durante o projeto.

Uma matriz de responsabilidades (MR) é usada para ilustrar as conexões entre um trabalho que precisa ser realizado e membros da equipe do projeto. Em projetos maiores, é possível desenvolver as MRs em vários níveis.

Por exemplo, uma MR de alto nível pode definir os grupos ou unidades da equipe do projeto responsáveis pelos componentes da EAP, enquanto MRs de nível mais baixo são usadas dentro do grupo para atribuir funções, responsabilidades e níveis de autoridade para atividades específicas (PMBOK, 2004).

O formato matricial, às vezes chamado de tabela, permite visualizar todas as atividades associadas a uma pessoa e também o contrário, ou seja, todas as pessoas associadas a uma atividade.

É comum utilizar um tipo de MR denominado gráfico RACI onde são atribuídas as seguintes funções aos membros da equipe: Responsável, Reporta-se, Consultoria e Informa (***R**esponsible, **A**ccountable, **C**onsult* e ***I**nform*, em inglês). Entretanto, conforme mostra a **Tabela 1** é possível adaptar a matriz para outras atividades, como por exemplo: Analisa (A), Modela (M), Desenvolve (D), Testa (T) e Instala (T).

PROJETO XYZ					
FASE	**EQUIPE**				
	Ana	**José**	**Paula**	**Pedro**	**...**
Análise					...
Elaboração de Casos de Uso	R	R	I		...
Elaboração de Diagramas de Classe	R	A			...
Modelo de Dados	I	C		R	...
...	...	...	...	...	...
Desenvolvimento					...
Módulo de Segurança	C	A, I	R		...
Cadastros Básicos		A	R		...
...	...	...	...	...	...

Tabela 1. Matriz de Designação de Responsabilidades do Projeto XYZ

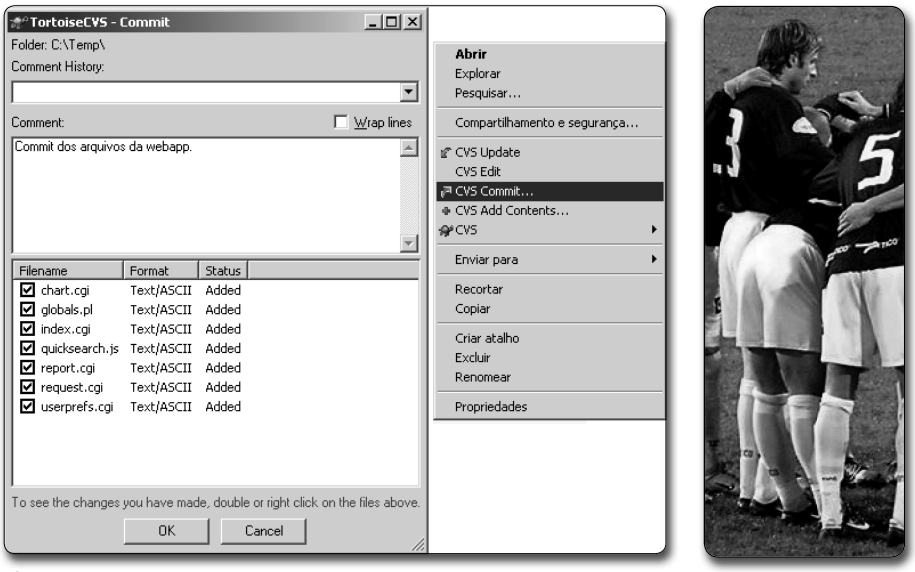


Figura 3. TortoiseCVS

Controle de Versões

Como já foi dito antes, a programação é a fase mais longa do desenvolvimento de um software. Por ser a mais longa, obviamente é a fase que consome mais recursos e também a mais sujeita a falhas, logo ela deve receber uma atenção especial por parte da gerência do projeto.

Esse artigo já dedicou duas seções sobre o código-fonte, a primeira tratava da importância da documentação e a segunda da padronização. Falta ainda tratar do controle de versões do código-fonte, para tanto existe o *Concurrent Versions System* (CVS).

O CVS é uma ferramenta *open source* que implementa as principais funções pertinentes ao processo de controle de versões, armazenando em seu repositório as modificações realizadas num arquivo ao longo do tempo.

Cada modificação é identificada por um número chamado *Revisão*. Toda *Revisão* armazena as modificações realizadas, quem realizou as modificações, quando foram realizadas, entre outras informações.

Qualquer *Revisão* poderá ser rastreada para fins de consulta, comparação ou união com outras *Revisões*. O CVS conta com um mecanismo capaz de controlar os acessos simultâneos e as modificações paralelas, garantindo a integridade das modificações e a atomicidade das operações.

Sem discutir em detalhes, o funcionamento do CVS se dá por meio da importação dos arquivos de um projeto para um repositório de arquivos. Cabe observar que as demais operações são compostas pela transferência de arquivos entre o servidor que abriga o repositório e o computador cliente.

Dessa forma, assim que o projeto for importado, o programador pode iniciar a transferência dos arquivos armazenados no repositório para a área de trabalho e realizar as modificações necessárias.

Tão logo essas modificações sejam concluídas, deverá realizar a entrega dos arquivos a fim de submeter as modificações realizadas sob o controle de versões, como pode ser visto no exemplo apresentado na **Figura 3**.

> **Nota:** para saber mais sobre o Tortoise CVS veja a edição 76 da ClubeDelphi.

O programador pode realizar a transferência dos arquivos armazenados no repositório que foram modificados para a sua área de trabalho por meio de uma operação de sincronização.

Por último, o CVS também oferece recursos para criar ramos de desenvolvimento paralelos, assim como, um mecanismo para designar um rótulo para identificar um conjunto de *Revisões* em determinado ponto no tempo.

A qualidade de um projeto de software é diretamente proporcional a qualidade dos processos adotados nas diversas fases do seu ciclo de vida. O controle de versões é visto como uma extensão natural do processo de desenvolvimento, permitindo que se possa paralelizar o desenvolvimento de forma coerente e padronizada, especialmente em se tratando de um conjunto de desenvolvedores.

Por essa razão, sem uma ferramenta ou metodologia adequada, esse controle torna-se extremamente complexo e improdutivo. É importante ressaltar que o CVS fornece uma abordagem consistente, controlada e flexível para gerenciar a evolução das mudanças, garantindo a integridade e a rastreabilidade dos arquivos modificados.

Entre as motivações para usar o CVS, pode-se destacar as seguintes:

- É uma ferramenta *open source*, madura e amplamente utilizada em projetos ao redor do mundo;
- Existe uma grande quantidade de documentação disponível na internet;
- É uma ferramenta multi-plataforma. Existem clientes e servidores do CVS portados para Windows, DOS, Linux, e outras.

Qualidade e Teste de Software

Para finalizar, cabe falar sobre a necessidade de garantir a qualidade do produto que está sendo entregue ao cliente. Desenvolver software de qualidade não é mais um requinte para poucos, transformou-se num fator de competitividade num mercado cada vez mais exigente.

O filósofo Nietzsche, no século passado, alertava: *Com o aumento da competição, a qualidade se torna mera propaganda. Vence aquele que melhor engana*. Essa receita é muito simples e fácil de seguir, todavia, quem tomar esse tipo de postura estará fadado ao fracasso.

Nos dias de hoje, a qualidade tornou-se requisito imprescindível para garantir a sobrevida de um software no mercado. Portanto, nesse contexto, podemos concluir que as empresas mais competitivas são as empresas que trabalham sob a ótica da melhoria contínua dos processos para aumentar a qualidade do processo de desenvolvimento e, conseqüentemente, aumentar a qualidade do produto final.

Estima-se que o custo decorrente da correção de um *bug* cresce dramaticamente à medida que ele é descoberto em fases mais adiantadas no processo de desenvolvimento. No entanto, ainda existe uma forte tendência nos desenvolvedores em negligenciar essa realidade e não dedicar o tempo mínimo necessário para a realização das atividades de qualidade e testes de software.

A necessidade de utilizar mecanismos de garantia de qualidade de software tem crescido à medida que as aplicações tornaram-se mais complexas. Esse crescimento tem sido exponencial em virtude da nossa dependência cada vez maior de aplicativos que controlam atividades de missão crítica; seja num hospital, na bolsa de valores ou no ônibus espacial.

Em termos práticos, isso significa que cada vez mais cedo as empresas estão introduzindo atividades de qualidade de software nas diversas fases do desenvolvimento. Mesmo os bons softwares são repletos de defeitos.

Para ilustrar, uma taxa de 0.51 defeitos a cada mil linhas de código é considerada aceitável pela indústria de TI. Ou seja, todos os softwares têm defeitos, no entanto, os softwares realmente bons têm poucos defeitos críticos não corrigidos; promovendo assim uma operação mais estável no ponto de vista do usuário final.

Mas afinal, nos softwares atuais, por que existem tantos defeitos? Poderíamos escrever um livro para tentar responder a essa pergunta e, ainda assim, talvez não seriam identificadas todas as causas possíveis. No entanto, pode-se elencar algumas causas como as mais prováveis, como as destacadas abaixo:

- Falta de comunicação entre os membros da equipe;
- A complexidade do software;
- Erros de programação;
- Mudança de requerimentos no meio do projeto e requerimentos ambíguos;
- Pressão da gerência para terminar o produto o mais rápido possível.

Revisões formais, estratégias de garantia de qualidade, planos de testes, gerência de riscos, entre outras atividades estão se tornando cada vez mais comuns no cotidiano das empresas. Em contrapartida, para dar sustentação a essa mudança de paradigma podemos notar a proliferação de ferramentas *open source* e comerciais para a execução e gerência das atividades de teste e qualidade de software.

Essa tendência também pode ser observada na evolução das suítes de desenvolvimento disponíveis no mercado. Tanto a Microsoft quanto a Borland estão apostando todas as fichas em suítes de desenvolvimento que suportam e fornecem ferramentas para todo o ciclo de vida do desenvolvimento, assim como ferramentas para a criação e gerência de testes funcionais, testes de carga, unidade, entre outros.

Conclusão

Esse artigo trouxe uma série de dicas e ferramentas, algumas delas baseadas em metodologias e outras obtidas na vivência profissional dos autores, que visam facilitar o desenvolvimento em equipe.

Procurou-se mostrar técnicas para todas as fases do ciclo de vida de um projeto de desenvolvimento de software, da análise aos testes, dando um maior destaque evidentemente para a programação. ●

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como trabalhar com o DUnit para testes unitários no Delphi.

www.devmedia.com.br/articles/viewcomp.asp?comp=540

Referências

PMBOK - Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos. Project Management Institute (PMI). 3ª ed. 2004.

SPÍNOLA, Rodrigo Oliveira. ARAÚJO, Marco Antônio P. UML na Prática: Construindo Diagramas de Classes. SQL Magazine, ed. 34, 2006.

Links

Home page do TortoiseCVS
www.tortoisecvs.org

DelphiDoc
www.softconsult.se/delphidoc.html

Web Mobile TechWeek

Realização: DevMedia GROUP

31 de Agosto e 01 de Setembro em São Paulo / SP

O Evento do desenvolvedor **Web e Wireless**

Palestras 100% técnicas!
Garanta sua participação e conheça o que as tecnologias Java e .net trazem de melhor para este mercado!

Para mais informações acesse:
www.devmedia.com.br/eventos/webmobiletechweek

Confira abaixo a Grade de Palestras:

Salas de Java

1º dia - 31 de Agosto

	Sala 1	Sala 2	Sala 3
9:00-10:20	Interfaces ricas na Web com Ajax - Parte 1	Padrões de Projeto Java EE - Parte 1	WEB 2.0: Conceitos e Tecnologias
10:30-11:50	Interfaces ricas na Web com Ajax - Parte 2	Padrões de Projeto Java EE - Parte 2	Mercado de Jogos para Celulares
12:00-13:20	Almoço	Almoço	Almoço
13:30-14-50	Mini Curso 1 - Java ME Primeiros Passos - Parte 1	Struts2 - A Evolução do framework - Parte	Utilizando SVG (Scalable Vector Graphics) com Java ME
15:00-16:20	Mini Curso 1 - Java ME Primeiros Passos - Parte 2	Struts2 - Desenvolvendo uma aplicação com ajax e validação - Parte 1	Otimização de Aplicações Java ME
16:20-17:00	BREAK	BREAK	BREAK
17:00-18:20	Além do básico: explorando APIs sofisticadas com Java ME nas plataformas Nokia - Parte1	Struts2 - Desenvolvendo uma aplicação com ajax e validação Parte 2	iReport/JasperReport - Criando relatórios para web
18:30-19:50	Além do básico: explorando APIs sofisticadas com Java ME nas plataformas Nokia - Parte2	JavaFX: interfaces em Java de forma fácil e portável - Parte 1	Google Web Toolkit passo-a-passo - Parte 1
20:00-21:20	Interfaces ricas para a Web com Flex 2.0 e Java	JavaFX: interfaces em Java de forma fácil e portável - Parte 2	Google Web Toolkit passo-a-passo - Parte 2

2º dia - 01 de Setembro

	Sala 1	Sala 2	Sala 3
9:00-10:20	Implementando uma loja virtual em Java EE 5 com JPA e EJB3 - Parte 1	Desenvolvendo um Game em Java ME - Parte 1	Planejamento e Execução de Testes em Aplicações Web - Parte 1
10:30-11:50	Implementando uma loja virtual em Java EE 5 com JPA e EJB3 - Parte 2	Desenvolvendo um Game em Java ME - Parte 2	Planejamento e Execução de Testes em Aplicações Web - Parte 2
12:00-13:20	Almoço	Almoço	Almoço
13:30-14-50	Implementando uma loja virtual em Java EE 5 utilizando JSF para a interface com o usuário.- Parte 1	Desenvolvendo um Game em Java ME - Parte 3	Java Wireless: O que nós precisamos saber sobre MIDP3?
15:00-16:20	Implementando uma loja virtual em Java EE 5 utilizando JSF para a interface com o usuário.- Parte 2	Introdução ao uso do bluetooth em J2ME	Mini Curso 2 – Conectividade em Java ME - Do básico ao avançado - Parte 1
16:20-17:00	BREAK	BREAK	BREAK
17:00-18:20	Desenvolvendo e implantando uma força de vendas em Palm - Parte 1	Desenvolvendo aplicações Web com o Spring Framework – Parte 1	Mini Curso 2 – Conectividade em Java ME - Do básico ao avançado - Parte 2
18:30-19:50	Desenvolvendo e implantando uma força de vendas em Palm - Parte 2	Desenvolvendo aplicações Web com o Spring Framework – Parte 2	Multimídia em J2ME RenatoFaria e eduardo peixoto

Salas de .NET

1º dia - 31 de Agosto

	Sala 1	Sala 2	Sala 3
9:00-10:20	Desenvolvimento para dispositivos móveis na plataforma .NET 2.0 - Parte 1	Linq - veja na prática os fundamentos deste projeto	Ciclo de vida de desenvolvimento com VSTS
10:30-11:50	Desenvolvimento para dispositivos móveis na plataforma .NET 2.0 - Parte 2	Mini-curso de C# - Parte 1	ASP.NET Caching
12:00-13:20	Almoço	Almoço	Almoço
13:30-14-50	Introdução ao ASP.NET AJAX	Mini-curso de C# - Parte 2	XAML
15:00-16:20	Aprenda na prática a criar um sistema de enquetes em ASP.NET - Parte 1	Acionamento de dispositivos eletro-mecanicos com Pocket PC	ASP.NET 2.0 na prática: Profiles e Membership
16:20-17:00	BREAK	BREAK	BREAK
17:00-18:20	Aprenda na prática a criar um sistema de enquetes em ASP.NET - Parte 2	Novidades no Windows Mobile 6	Criação de um portal com ASP.NET 2.0 e WebParts
18:30-19:50	Desenvolvendo programa para PocketPC com rede sem fio e WebServices - Parte 1	Introdução ao Silverlight	Desenvolvendo Gadgets Corporativos para Windows Vista
20:00-21:20	Desenvolvendo programa para PocketPC com rede sem fio e WebServices - Parte 2	Criando aplicações WAP com ASP.NET	.net 3.0 - Xaml Browser Applications

2º dia - 01 de Setembro

	Sala 1	Sala 2	Sala 3
9:00-10:20	Aplicações com ASP.NET 2.0 e AJAX - Parte 1	Otimizando aplicações em Windows Mobile 5.0	Aplicações Web para dispositivos móveis
10:30-11:50	Aplicações com ASP.NET 2.0 e AJAX - Parte 2	Construindo uma aplicação Mobile passo a passo com Visual Studio 2005	Linq para Compact Framework
12:00-13:20	Almoço	Almoço	Almoço
13:30-14-50	Construa uma aplicação 100% OO com ASP.NET - Parte 1	Envio e recebimento de dados no Pocket usando FTP	Silverlight na prática
15:00-16:20	Construa uma aplicação 100% OO com ASP.NET - Parte 2	Windows Mobile no Windows CE	WorkFlow Foundation no ASP.NET
16:20-17:00	BREAK	BREAK	BREAK
17:00-18:20	AJAX Control Toolkit	SQL Server 2005 Compact Edition - Parte 1	Desenvolvimento WEB: Uma Abordagem Utilizando OO e Técnicas de Teste de Software
18:30-19:50	Net Framework 3.0 - Conheça o WPF e surpreenda-se	SQL Server 2005 Compact Edition - Parte 2	Testes automatizados em aplicações ASP.NET

Patrocínio:

NOKIA Connecting People

Caelum Ensino e Soluções em Java www.caelum.com.br

Apoio Cultural:

TEMPO REAL LIVRARIA DE INFORMÁTICA E NEGÓCIOS

Apoio:

MOBILEZONE · DFJUG · mcdbabrasil · INFORMÁTICA www.ramosdainformatica.com.br · Oficina da Net · TreoChé · codigofonte.net · aspbrasil · <criarweb> · 10 anos Cobre Bem

Cursos Online

Assinatura

ClubeDelphi PLUS

Mais conteúdo DELPHI por muito menos!

A Revista **ClubeDelphi** oferece para seus assinantes uma série de **Cursos Online** de alto padrão de qualidade . Conheça abaixo os cursos já disponíveis..

Curso em destaque

Aplicações Client/Server com dbExpress e Firebird

Confira neste curso online como criar uma aplicação client/server completa no Delphi 7, utilizando dbExpress e Firebird.
Aprenda também: Como trabalhar com um driver dbExpress específico e gratuito para o Firebird ; Como utilizar Orientação a Objetos, e técnicas como herança visual de formulários e relatórios ; saiba como colocar regras de negócios no banco de dados, usando Triggers e Stored Procedures; E crie relatórios com o Quick Report, Rave Reports e Report Builder, e muito mais.

Confira o plano de aula completo:
www.devmedia.com.br/clienteserver

A sua melhor opção de aprendizagem!
Assine a **ClubeDelphi** e comece já seu treinamento!
www.devmedia.com.br/assine

Outros cursos disponíveis: **www.devmedia.com.br/curso**

- Curso Online - Sistema SysCash
- Aplicações client/server com Windows Forms no Delphi 2006
- Criando uma Aplicação multi-camadas Completa com Delphi
- Aplicações WEB com IntraWeb e Delphi 7 (Delphi Win32)

DevMedia GROUP **Mais Informações: www.devmedia.com.br/central - Tel.: 21 2220-5375 / 2220-5435**